April
2018

Early Rumour Detection in Social Media using Machine Learning

PSG 16 – DR ZHIWEI LIN – BENG SOFTWARE ENGINEERING

THOMAS FRANKLIN B00607399

Abstract

Within the last 2 years social media sites such as Twitter has noted a rise in what is known as 'fake news'. Fake news can be described as the deliberate spreading of misinformation, published under the guise of authentic news to influence the readers thoughts and is becoming a much larger issue in recent times with the rise of social media and particular “influencers”.

As social media tends to be an area where users can express their own belief, large influencers can use it as a playground to misinform their followers and hence start the process of spreading fake news. As content on social media can reach hundreds of thousands of users in minutes, which means current methods of combating fake news is often to slow to stop the spread before it starts, it was time to look into solutions which can aid in detecting fake news quicker and prevent the spread before it is too late.

The aim of this project is to use machine learning to classify all public Tweets from Twitter as a rumour or non-rumour where the results could then be retrieved through a rich internet application for visualisation. From research conducted there have been very few existing projects that have tackled rumour detection in social media, and from the information gathered this will be the first project which uses natural language processing and machine learning for the early detection of a rumour in social media.

The final product produced is a service which continually streams Tweets from Twitter based on keywords, processes them and classifies them as either as rumour or non-rumour which is then stored in a SQL database for later retrieval on an interactive web service which will show a breakdown of the results in an easy to understand format using visual material.

The use of natural language processing and machine learning is only the start of tackling fake news, on its own it reduces the amount of results which will still need accessed. As the previous methods required individuals to cross-verify everything. With the successful completion of this project, it will automate some of the process which means the results can be found earlier in the process.

Acknowledgements

This project would not have been possible without the support of my mentor Dr. Zhiwei Lin for his guidance and support throughout. Another thanks goes towards the Ulster University teaching staff of whom I have had the pleasure to learn from over the last 4 years, and finally I would like to thank those I had the pleasure of working with while on placement – most importantly Aaron Long who was a mentor to me when starting out in placement and with his continued support I have become a better developer. More importantly my final acknowledgement goes to my beloved girlfriend Samantha who was a constant stream of support throughout, and without her support the project would not have been completed.

Table of Contents

1. Introduction

This report will provide details on the solution developed to tackle the problem identified in the report, along with describing the development process of the proposed solution along with the management and elicitation of the product and user requirements which will be documented in the report, the report will also highlight the design and testing which had been completed to ensure that the project is a success.

1.1 Problem Elucidation and Statement

Fake news[1] is a term which has risen in popularity within the last few years, especially when it comes to social media sites such as Twitter and Facebook (Meyer, R, 2018; Science | AAAS, 2018), although in recent months there has been responses from Facebook on their plans to tackle fake news (The Guardian, 2018; Bloomberg.com, 2018), Twitter on the other hand has been less vocal in their attempts and have only suspended accounts and emailed those who may have been in contact (Ecommerce Week. 2018).

Evidence has come to light in the last few weeks about how the 2016 US Presidential Elections were influenced by what is termed fake news, and how the Russians pioneered and used it to influence the election (Vox, 2018; Conger K, 2018), had there been detection measures in place the spread of the misinformation could have been stopped earlier to minimise the impact on the results.

If nothing is done to try and reduce, or completely stop the spread of misinformation then it will continue to sabotage future important events. Out of the two main social media sites, only Facebook has begun to address the issue, this highlights a weakness in Twitter which must be addressed in order to implement a solution in the early detection of rumours minimising risks.

Should the project be capable of aiding in the early detection of social media the project code should be reusable for different functions that require assistance in the detection of rumours, which gives it large commercial value as any company could reuse the system with their own filters to explore users opinion on their company and if users are spreading misinformation about their products which could be brand damaging they would be able to intervene early.

1.2 Project Aim

This project aims to aid in the early detection of Tweets from Twitter, in order for the appropriate action to be taken to prevent the spread of misinformation; often referred to as fake news. With the use of machine learning, this project will continuously stream Tweets from Twitter in real-time with the aid of Twitters API and the use of configuration filters for areas of interest, such as Politics, Crisis', Disasters etc.

The project will then classify the individual Tweets as a rumour or non-rumour using the trained classifier where the results will be stored in a database which allows for an interactive website which can get the data required to present the information using various visualisation techniques.

1.3 Project Objectives

[1] false, often sensational, information disseminated under the guise of news reporting (Fake news definition and meaning | Collins English Dictionary, 2018)

Table 1 Project objectives table

Id	Objective
1	Design and implement a system which can continuously process Tweets from Twitter using configurable filters
2	Implement a classifier capable of assigning a rumour or non-rumour label based on the text from a Tweet
3	Design and implementation of a website to display the information in meaningful ways using best practices
4	Design and implementation of a backend service which can retrieve the data
5	Follow a project plan to ensure completion of the project by the project deadlines
6	Evaluate different machine learning libraries which can be used for the development of the classifier in objective 2
7	Evaluate different frameworks and libraries which can ensure a well-executed system which adheres to appropriate system design considerations
8	Implement a system which is capable of running independently without much user interaction to adhere to the real-time aspects required of the project aim
9	Design and implement a system which can be reused by others who wish to monitor rumours of a given topic

1.4 Selected Software Lifecycle Methodology

The lifecycle followed by this project was the Extreme Programming (XP) software development methodology.

2. Requirement Control Document

This section will document the methods used by the project owner to gather user/product requirements, along with tables outlining the final set of functional and non-functional requirements which had been identified with the prioritisation value using the prioritisation technique known as Wiegers Relative Weighting, as well as the explanation as to how the requirements evolved during the lifecycle of the project.

2.1 Requirements Gathering Methodology

The initial requirements of the project and for any additional requirements were gathered using the following techniques: peer reviews, scenarios and walk-throughs which were all performed with various members of the stakeholders of the project. The techniques suited the format of the project, as regular meetings were held with the stakeholders in the early stages, and through these meetings the requirements could be identified using the three techniques mentioned above and they were refined using brainstorming techniques to help shape them in to tangible requirements.

2.2 Final Requirements

For the prioritisation of requirements, the project utilised Wiegers Relative Weighting technique (Wiegers K, 1999) which is a technique which uses cost, value and risk to prioritise the requirements. Table 2 shows the final set of functional requirements for the developed system along with their prioritisation value, while in Table 3 it shows the non-functional requirements of the system along with their prioritisation value.

The prioritisation value is calculated using Wiegers Relative Weighting formula as below:

$$Priority = \frac{Value\%}{Cost\%\ x\ CostWeight + Risk\% * RiskWeight}$$

The higher the value the higher the priority, tables 2 and 3 only show the value, they are organised by the Requirement ID rather than the prioritisation value.

Table 2 Final Functional Requirements and Prioritisation Value

ID	Description	Prioritisation Value
FR.01	The system shall assign a rumour or non-rumour label to a processed Tweet	0.46
FR.02	The system shall provide details on if it is running	0.19
FR.03	The classifier shall only be able to assign a rumour or non-rumour label	0.94
FR.04	The system shall be able to link back a Tweet to a particular user	0.58
FR.05	The system shall be able to link back a Tweet to a particular hashtag	0.58
FR.06	The system shall be configurable with the filter list	0.47
FR.07	The system shall be able to report on successes/failures	0.18
FR.08	The system shall remove/discard any items that are not a Tweet object	0.24
FR.09	The system shall only handle Tweet objects from the queue reader	0.24
FR.10	The system shall be able to receive user requests	0.44
FR.11	The system shall be able to display status information when user navigates to homepage	0.35
FR.12	The system shall return responses in JSON format	0.52
FR.13	The system shall provide help to the users	0.67
FR.14	The system shall allow users to report particular terms	0.32
FR.15	The system shall keep up with the real time Tweet stream	0.83

Table 3 Final Non-Functional Requirements and Prioritisation Value

ID	Description	Prioritisation Value
NF.01	The system shall be configurable to account for the needs of the system	0.16
NF.02	The system shall make use of a two-stage classification process	0.47
NF.03	The system shall accurately store the data to the database	0.55
NF.04	The system shall only add Tweets that are not a retweet to the Queue	0.55
NF.05	The system shall be robust enough to restart on failure	0.58
NF.06	The system shall ensure that the message is valid from the queue	0.3
NF.07	The system shall only display relevant information to the status of the system	0.35
NF.08	The system shall be free from SQL injection attempts	0.6
NF.09	The system shall be secure	0.32
NF.10	The system shall be able to respond to user requests within 15 seconds	0.55
NF.11	Data requests from the Frontend shall be executed directly to the API	0.24
NF.12	The system shall be pleasing to the eye	0.38

2.3 Requirements Evolution

From the initial stages of the project, FR.3 and FR.15 have remained as they are vital requirements which the project must meet, while the others found in tables 2 and 3 are a result of brainstorming with the stakeholders as outlined in section 2.1 which transformed some of the original requirements which were considered incomplete in to more meaningful and tangible requirements.

As the project followed an agile lifecycle methodology there was plenty of opportunity to revisit the requirements, and add to them if required; as each iteration of the system went through various stages, one of which was release planning/sprint planning where user stories were prioritised, and as part of planning a user story the requirements had to be taken in to account as to what is being addressed – during these stages, requirements had been amended, removed or split to be less ambiguous.

3. System Overview

This section will outline the system design, which includes the architecture of the system. The system was designed with performance in mind as it has to continuously stream and process Tweets in real-time while not hindering the user performance which is why the system was split in to multiple micro-services, which meant that each service only had one responsibility which allows for easy scalability of the service if required, and as the services are loosely coupled[2] they can be maintained easier.

Along with the system design, the section will also cover the User Interface design, which will include the wireframes which have been designed to aid with development, and the considerations made for design best practices such as HCI[3]. As the system uses a SQL database for the storage and retrieval of classified

[2] Loosely coupled means that you can update the services independently; updating one service doesn't require changing any other services. (NGINX, 2018)

[3] Human Computer Interaction - a field of study focusing on the design of computer technology and, in particular, the interaction between humans (the users) and computers (UX Courses, 2018)

Tweets, section 3.3 will show the design of the database, displayed using ER diagrams. Section 3.4 provides information on the system through the uses of use case diagrams and activity flow diagrams and aims to provide detail on how users will interact with the system.

3.1 System Design

The system was designed with micro-services in mind, to ensure that the system can easily be scaled if required and also to ensure that the system can deal with the real-time aspects of the service. With all this in mind, the system was split in to two parts, one which dealt the with processing, and then one which dealt with the user interaction – this allows the system to be more robust in the fact that users interactions will not slow down/affect the processing of data, while the processing of data will not negatively impact the user. Figure 1 shows the architecture of the system where it highlights the high-level components of the system.

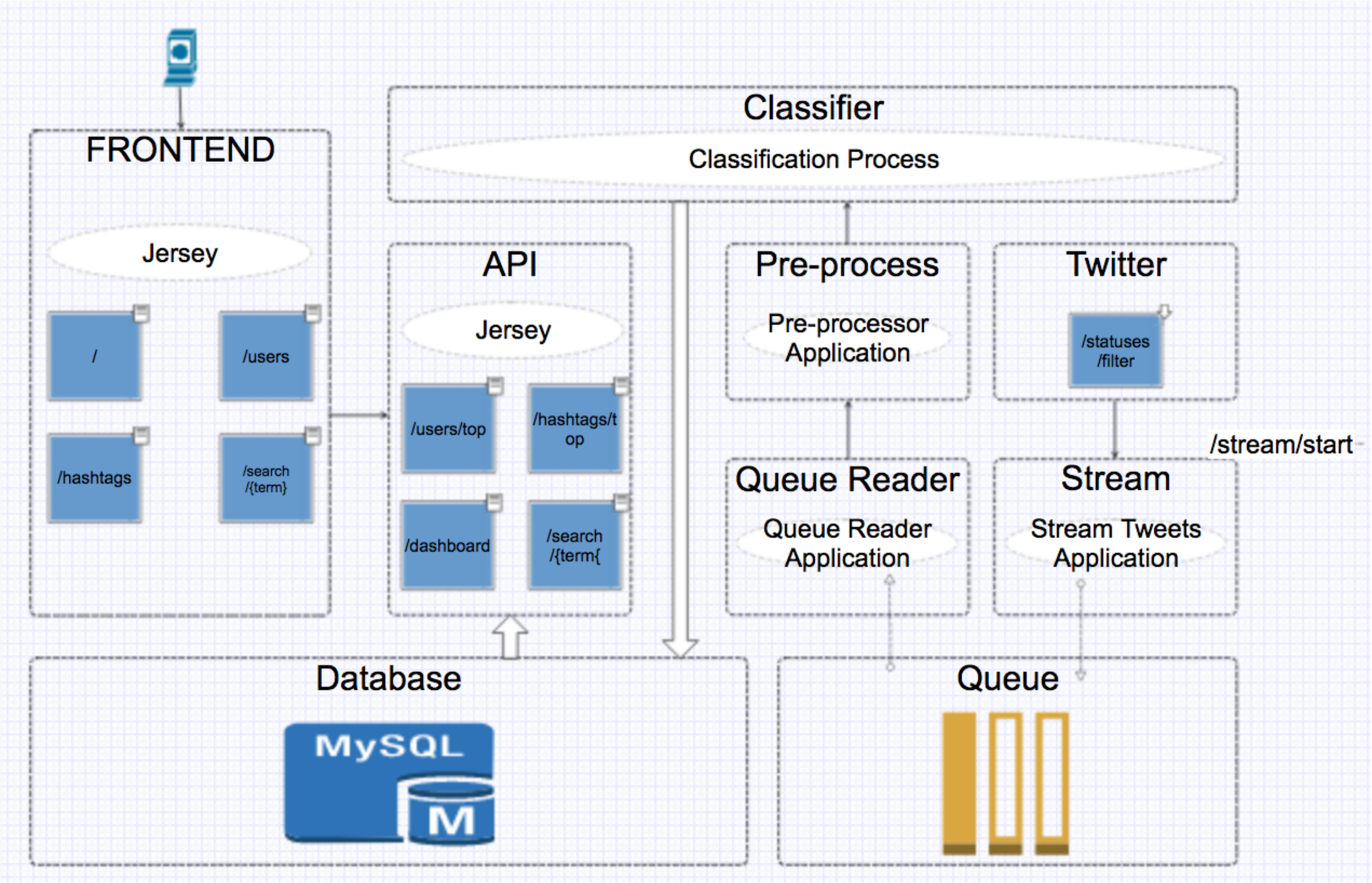


Figure 1 High-level System Architecture Diagram

From figure 1 each of the core components are listed, beginning with the one labelled *Twitter* is the Twitter API, and the *Stream* calls the API using Twitter4J[4] which opens up a filtered stream[5], the filter is a list of words which will be streamed in to the service, and is one of the ways in which the system can be reused as different companies can apply their own filters for topics which are relevant to them.

The *Stream* service is the control over the system, and has an endpoint exposed to start and stop the stream when required, when the stream is started it will continuously stream Tweets as per the product requirements identified in section 2.2.

[4] http://twitter4j.org/en/index.html – Twitter4J is an unofficial Java library for the Twitter API. (Twitter4J, 2018)

[5] https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter.html - Returns public statuses that match one or more filter predicates. (POST statuses/filter — Twitter Developers. 2018)

From the stream, each individual streamed Tweet will be loaded on to a message queue in order to keep up with the volume of Tweets coming through the service, the service labelled *Queue Reader*, reads from the same message queue and dispatches it to the *Pre-process* service – where some pre-processing is done on the Tweet, such as lowercasing, removal of certain symbols, etc. in preparation for the classification process, using an endpoint exposed on the service labelled *Classifier* the pre-processor posts the processed Tweet item to it, this is where the Tweet body gets assigned a label as either a rumour or non-rumour based on the classification process outlined in section 3.5.1 – after the classification process, the *Classifier* saves the item to the *Database*, of which the design is outlined in section 3.3.

This is the end of what is referred to as the offline side of the service, which continuously processes in the background without user interaction, moving on to the service labelled *API* is where the user interacts with the service and is referred to as the online part of the service, and it will have various endpoints to retrieve different bits of data in a RESTful[6] way. The service labelled *Frontend* works closely with the *API* as the *Frontend* makes use of the APIs endpoints to retrieve the required data from the *Database* to display to the user on the *Frontend*.

All the services identified in Figure 1 are all ran within their own Docker containers, as a container is a lightweight, standalone package which have all the required tools to run the service (Docker, 2018), i.e. the *Frontend* service will have Tomcat[7] running which will expose the ports in order to connect to the web service, while *Classifier* will have Weka[8] and Mallet[9] running in order to perform the classification of the Tweets.

3.2 Interface Design

The following section will provide the required evidence to show that the interface design was done with best practices in mind by following the core principles of HCI and how these considerations will help to improve the usability and accessibility of the final product. Along with the evidence of following the best practices, it will also provide the wireframes that were produced as a guideline for the finalised pages of the website.

3.2.1 HCI Considerations

When it comes to HCI, there are two key practices which typically come up, the 10 Usability Heuristics for User Interface Design (Jakob Nielsen, 1995) and the 8 Golden Rules of Interface Design (Ben Shneiderman's, 1986). The 10 Usability Heuristics iterate over the 'golden rules' (UX Courses, 2018) and as such the project has used these heuristics when considering the design of the interfaces in the project, table 4 shows the 10 heuristics with an explanation.

These 10 heuristics set the guidelines which have been followed when considering the interface of the project, and as such they aim to enhance the usability and accessibility of the service. Interface design should not end at the website, as the project relies heavily on the backend API, the interface of the API needs to be considered, which is why HCI has to be

[6] A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. (SearchMicroservices, 2018)

[7] The Apache Tomcat® software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies. (Apache Tomcat Project, 2018)

[8] Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. (Weka 3 - Data Mining with Open Source Machine Learning Software in Java, 2018)

[9] MALLET is a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text. (MALLET, 2018)

applied through the project to ensure that all aspects applies the principles set out by Jakob Nielsens 10 Usability Heuristics for User Interface Design.

Table 4 10 Usability Heuristics for Interface Design and a brief explanation

ID	Heuristic	Explanation
1	Visibility of system status	Keep users informed of system progress, i.e. use of progress bars, etc.
2	Match between system and real-world	Use language familiar to the user, i.e. use icons that represent real-world things
3	User control and freedom	Allow a "quick escape", i.e. if user finds they've went too far, allow them to go back easily
4	Consistency and standards	Follow conventions and don't use multiple words/icons to mean the same thing, i.e. use words consistently
5	Error prevention	Avoiding situations which could cause an error is better than descriptive errors, i.e. having a fault free system
6	Recognition rather than recall	Minimising users' memory load, i.e. the user shouldn't have to recall something from a previous page if the information is used elsewhere
7	Flexibility and efficiency of use	Allow users' control over the system to meet their needs
8	Aesthetic and minimalist design	Only present users with information that is relevant, do not overburden them with too much
9	Help users recognise, diagnose, and recover from errors	If an error occurs, use common terminology and do not hide behind error codes
10	Help and documentation	Although it is best to develop a system which requires minimal support, if there is documentation required ensure it is easy to navigate and gives concrete steps

3.2.2 Wireframes

Wireframes are used early in a project to establish basic page structures before any visual design or content is added (Experience UX, 2018). As a wireframe is only an early visual to provide an indication of where core page content and functionality should be positioned – not all structural elements will be shown in them, and as such they are only guidelines for the final design.

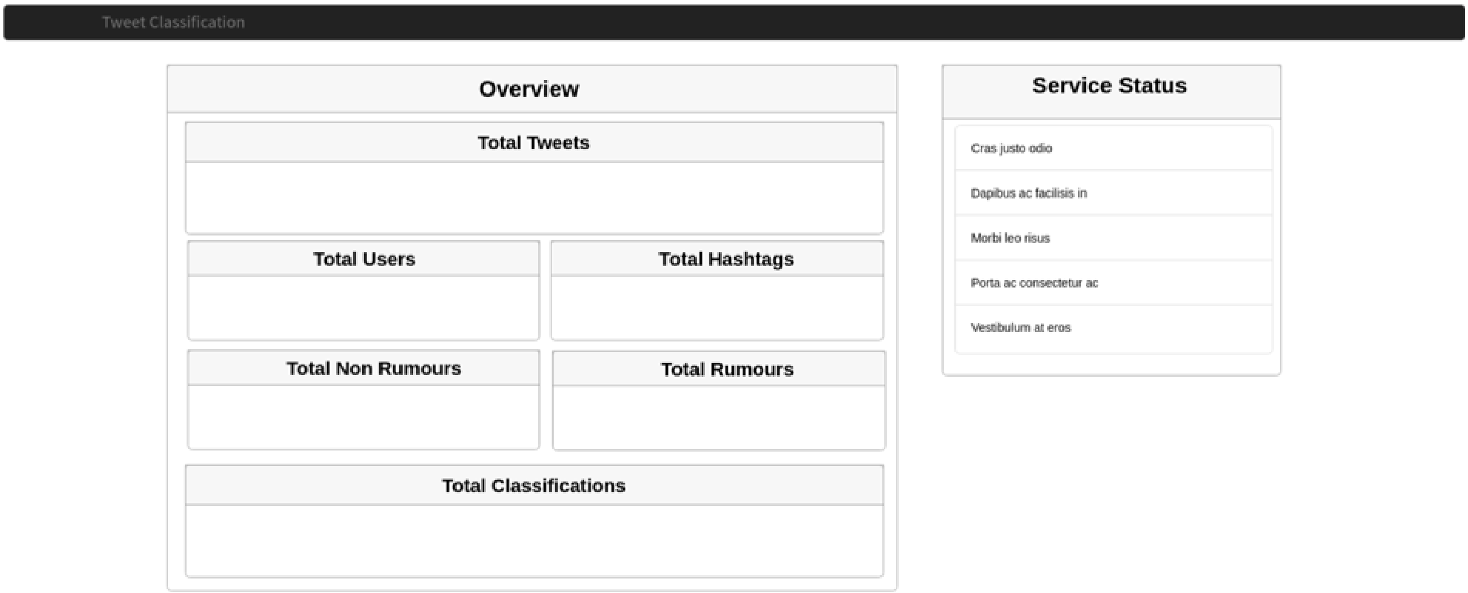
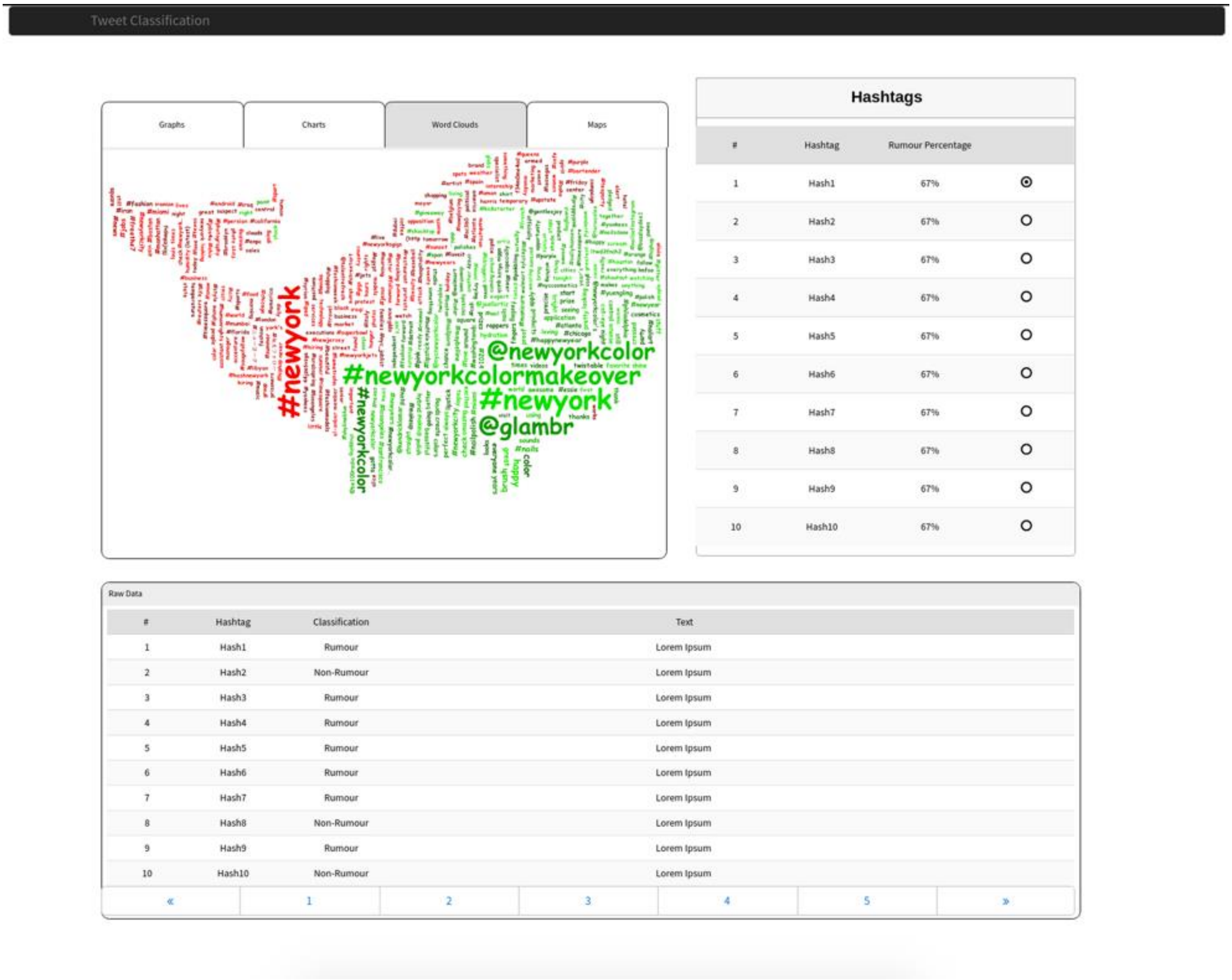


Figure 2 Wireframe of the Dashboard - the "homepage" of the website

Figure 2 is a wireframe of the dashboard of the service, which gives an overview of the data which is stored and gives the user a quick glance indication of the system status, such as which services are running – which relates back to heuristic 1 from table 4, about the guideline for providing users with information on the systems current status.

In Figure 3 it illustrates what key information could be provided when the user goes to the "hashtags" section of the service, and as such will allow the user to select different hashtags based on their rank and retrieve information specific to that hashtag – using the term "hashtag" corresponds with heuristic 4 from table 4, as the term "hashtag" is what users' of the service will be familiar with.



Hashtags

#	Hashtag	Rumour Percentage	
1	Hash1	67%	◉
2	Hash2	67%	○
3	Hash3	67%	○
4	Hash4	67%	○
5	Hash5	67%	○
6	Hash6	67%	○
7	Hash7	67%	○
8	Hash8	67%	○
9	Hash9	67%	○
10	Hash10	67%	○

#	Hashtag	Classification	Text
1	Hash1	Rumour	Lorem Ipsum
2	Hash2	Non-Rumour	Lorem Ipsum
3	Hash3	Rumour	Lorem Ipsum
4	Hash4	Rumour	Lorem Ipsum
5	Hash5	Rumour	Lorem Ipsum
6	Hash6	Rumour	Lorem Ipsum
7	Hash7	Rumour	Lorem Ipsum
8	Hash8	Non-Rumour	Lorem Ipsum
9	Hash9	Rumour	Lorem Ipsum
10	Hash10	Non-Rumour	Lorem Ipsum

Figure 3 Top Hashtags Wireframe

As the service groups the rumours and non-rumours between users and hashtags, the wireframe in figure 3 also covers the "users" classification page where it too will show the top users rank and allow the user to see various visualisation techniques for the user data. As the classification is done on an individual Tweet, the users/hashtags are just how the data is grouped, and as such the pages for "users" and "hashtags" should be visually similar in order to adhere to heuristic 4 and 6 from table 4.

3.3 Data Support Design

This section will aim to cover the consideration of data validation and security measures for the data of the project, along with the database design artefacts, such as the ER Diagram. As this project is very focused on the accuracy of information and the speed in which it is retrieved, getting the design of the database right is important, as a poorly design database will negatively impact the performance of a system which relies on data (Database Trends and Applications, 2018; SearchDataManagement, 2018).

3.3.1 Consideration for Security and Data Validation

The most common attacks on a database fall in to the following categories; SQL Injection, Buffer Overflow vulnerabilities, Denial of Service, Privilege Escalation and Weak Authentication (Checkmarx, 2018) all of which can be avoided with good database design and few basic steps to avoid such attacks. First of all, to combat privilege escalation, creating a user and revoking all access is performed, and then only granting the required privileges to the database and tables;

```
REVOKE ALL PRIVILEGES ON * FROM 'twitter'@'%';
```

(Revoking all privileges for everything for user ‘twitter’)

```
GRANT SELECT, UPDATE, DELETE, INSERT ON twitter_classification.* TO 'twitter';
```

(Only granting SELECT, UPDATE, DELETE and INSERT access to twitter_classification database for user ‘twitter’)

By removing privileges to all other databases, the user ‘twitter’ will not be able to escalate their privileges to get more access than they are allowed. Secondly for weak authentication, using passwords of suitable length and complexity will be applied, and ensuring that the ‘root’ user has a complex password set.

As Denial of Services (DoS) attacks on a database are often cause by Buffer Overflows (Checkmarx, 2018) and are designed to halt the database, by putting rate limits on the ‘twitter’ user will help to prevent a DoS attack and buffer overflows;

```
ALTER USER 'twitter'@'%' WITH MAX_QUERIES_PER_HOUR 5000;
```

(Setting the maximum queries per hour for user ‘twitter’)

Finally, for SQL Injection, this will be addressed from the API level, where no data will directly be inserted to the database as all communication with the database will be executed as a Prepared Statement.

Addressing the validity of data should be performed at all levels to ensure a smooth user experience, i.e. a user should not have to wait till they submit something to find out that there is an issue with what they entered. Although in terms of database design, data validation can be done by using a few simple techniques (Validation | Databases | ICT, 2018);

1) Type – using appropriate field types, i.e. for fields which will only contain numbers, use a numeric type, integer, big integer etc.
2) Presence – for compulsory fields, ensure that the database design acknowledges that they cannot be null or empty
3) Uniqueness – any data which should only occur once should be made as a unique field, or a combination of fields should be unique
4) Range – limit the amount of data which it can hold, i.e. if you know the text will be a maximum of 300 characters, then only allow it to store that amount
5) Format – particular inputs might need to be in a certain format

The list above is some of the techniques which can be applied in database design to ensure that data is validated correctly and in the appropriate format, which is expected by the application.

3.3.2 *Database Design and Related Artefacts*

The project went with a SQL database as the data has strong relations between a Tweet and a Classification value which then had relationships between individual users of Twitter and could have numerous hashtags that were in the original Tweet.

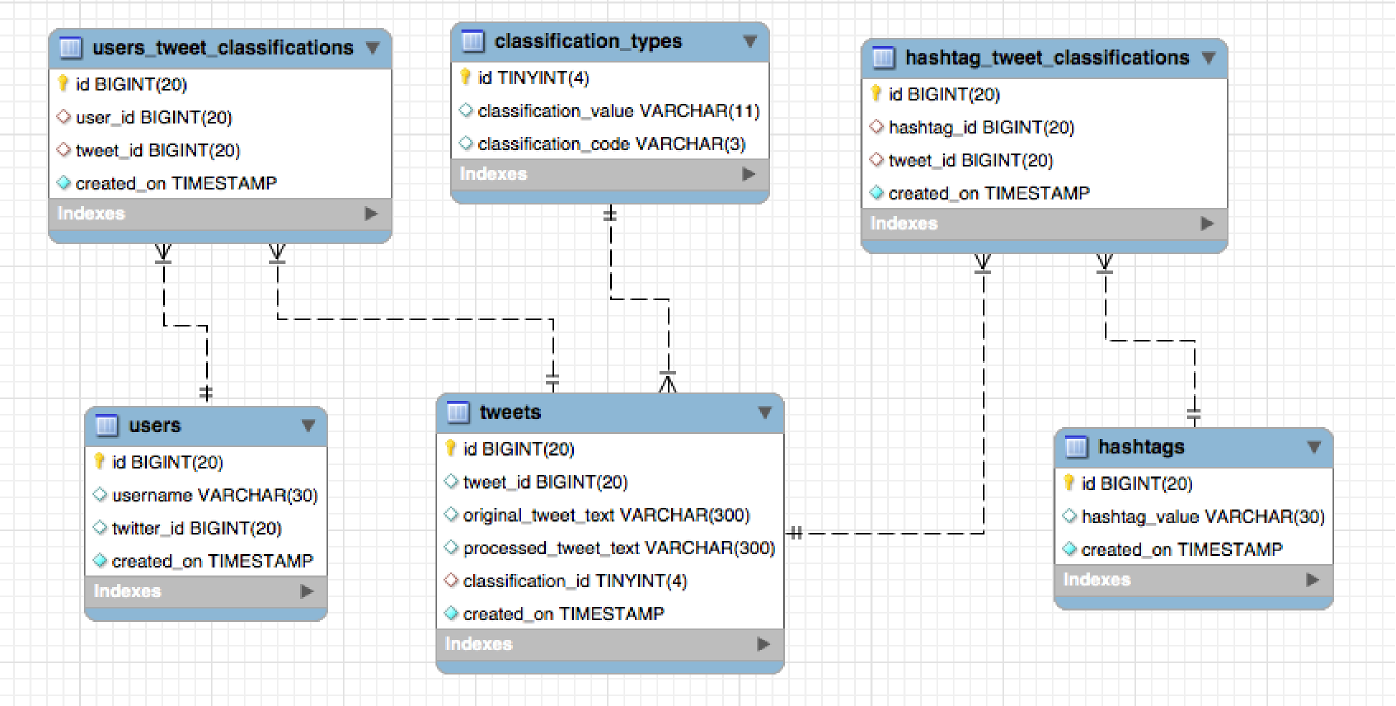
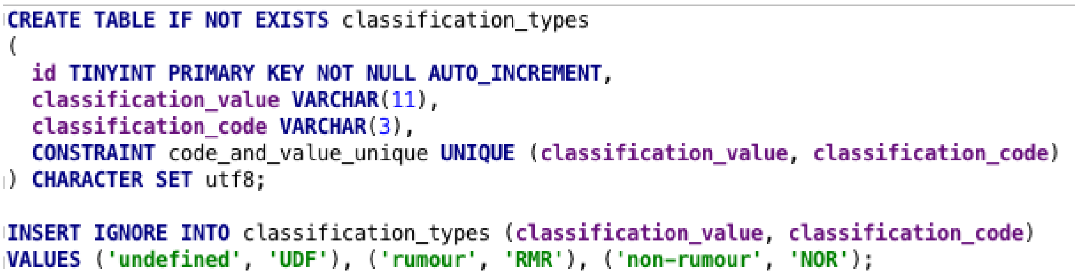


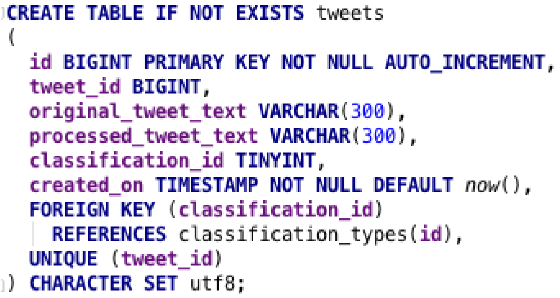
Figure 4 EER Diagram of twitter_classification Database

From figure 4 which illustrates the EER (Enhanced Entity-Relationship) diagram of the database, the relations are indicated by the dashed lines and the cardinality between tables is indicated at the end of the lines. With the EER Diagram in figure 4 it highlights how the database could be queried to get the Tweets and classification values for a particular hashtag or user of Twitter; which is important as this is the type of information which will be needed to present to the users of the service.



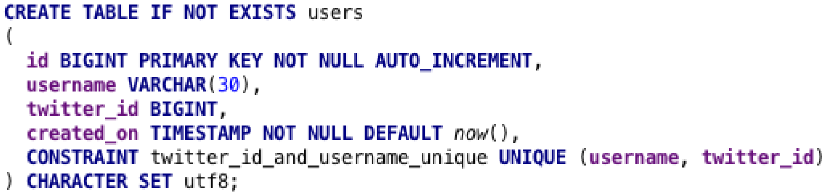
```
CREATE TABLE IF NOT EXISTS classification_types
(
  id TINYINT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  classification_value VARCHAR(11),
  classification_code VARCHAR(3),
  CONSTRAINT code_and_value_unique UNIQUE (classification_value, classification_code)
) CHARACTER SET utf8;

INSERT IGNORE INTO classification_types (classification_value, classification_code)
VALUES ('undefined', 'UDF'), ('rumour', 'RMR'), ('non-rumour', 'NOR');
```

CODE 1 Create Table Statement for classification_types



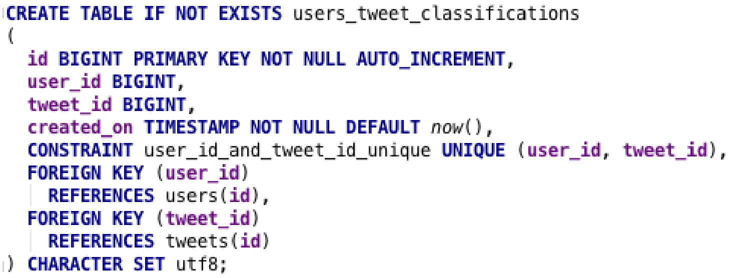
```
CREATE TABLE IF NOT EXISTS tweets
(
  id BIGINT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  tweet_id BIGINT,
  original_tweet_text VARCHAR(300),
  processed_tweet_text VARCHAR(300),
  classification_id TINYINT,
  created_on TIMESTAMP NOT NULL DEFAULT now(),
  FOREIGN KEY (classification_id)
    REFERENCES classification_types(id),
  UNIQUE (tweet_id)
) CHARACTER SET utf8;
```

CODE 2 Create Table Statement for tweets



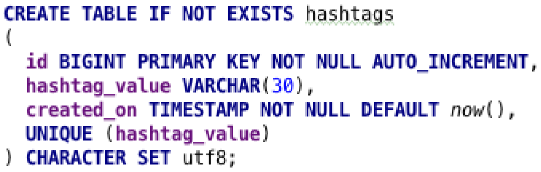
```
CREATE TABLE IF NOT EXISTS users
(
  id BIGINT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  username VARCHAR(30),
  twitter_id BIGINT,
  created_on TIMESTAMP NOT NULL DEFAULT now(),
  CONSTRAINT twitter_id_and_username_unique UNIQUE (username, twitter_id)
) CHARACTER SET utf8;
```

CODE 3 Create Table Statement for users



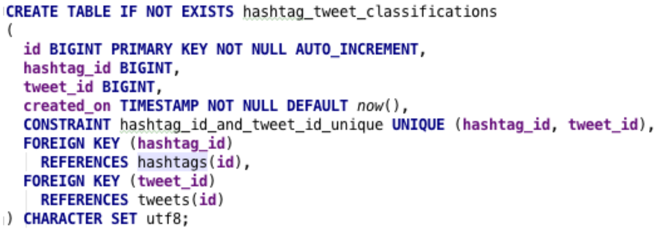
```
CREATE TABLE IF NOT EXISTS users_tweet_classifications
(
  id BIGINT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  user_id BIGINT,
  tweet_id BIGINT,
  created_on TIMESTAMP NOT NULL DEFAULT now(),
  CONSTRAINT user_id_and_tweet_id_unique UNIQUE (user_id, tweet_id),
  FOREIGN KEY (user_id)
    REFERENCES users(id),
  FOREIGN KEY (tweet_id)
    REFERENCES tweets(id)
) CHARACTER SET utf8;
```

CODE 4 Create Table Statement for users_tweet_classifications



```
CREATE TABLE IF NOT EXISTS hashtags
(
  id BIGINT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  hashtag_value VARCHAR(30),
  created_on TIMESTAMP NOT NULL DEFAULT now(),
  UNIQUE (hashtag_value)
) CHARACTER SET utf8;
```

CODE 5 Create Table Statement for hashtags



```
CREATE TABLE IF NOT EXISTS hashtag_tweet_classifications
(
  id BIGINT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  hashtag_id BIGINT,
  tweet_id BIGINT,
  created_on TIMESTAMP NOT NULL DEFAULT now(),
  CONSTRAINT hashtag_id_and_tweet_id_unique UNIQUE (hashtag_id, tweet_id),
  FOREIGN KEY (hashtag_id)
    REFERENCES hashtags(id),
  FOREIGN KEY (tweet_id)
    REFERENCES tweets(id)
) CHARACTER SET utf8;
```

CODE 6 Create Table Statement for hashtag_tweet_classifications

The code entries 1 through to 6 illustrates how the tables have been constructed along with the types of data each field accepts and the constraints which have been applied, code 1 is the statement which had been executed to create the *classification_types* table which stores the two types of classification values which the system deals with; rumours and non-rumours, hence why it is created and then the *INSERT* statement is used to insert values for *undefined, rumour, and non-rumour* – undefined is a precautionary measure for instances which may have thrown an error during classification.

For the fields used, for *classification_code* it is a field with a range of 3 characters as this is all the is required, *classification_value* on the other hand has a range of 11 characters, which is one more than the largest character size of *non-rumour* and a constraint has been placed for the uniqueness of *classification_code and classification_value*, in order to ensure that there is only one entry for each classification type.

Code 2 illustrates the code executed to create the *tweets* table, which is where individual Tweets will be inserted, *original_tweet_text* is the original content of the Tweet without any processing done, and the *processed_tweet_text* is the text which went through the classification process and had processing applied, both of which have a field type of VARCHAR(300) which means they can only store 300 characters, and as the maximum size of a Tweet is 240 characters, this is enough space for them to be stored. From the code it indicates that there is a foreign key for *classification_id* to *classification_types.id* which also iterates what is indicated in the ER diagram on figure 4. As each *tweet* only gets assigned one classification the *tweet_id* is unique in order to ensure that the same Tweet does not get processed more than once which is part of the data validation that is performed at the database level.

For the table *users* the creation is illustrated in code 3, and will store the *username* which is the screenname that the User of Twitter is known as, with a character limit of 30 set, while the combination of *twitter_id* and *username* requires uniqueness to ensure that a particular users Tweets can be traced, as two users could have the same screen name, but two users will not have the same *twitter_id and username*.

The *users* table is related to an individual Tweet through the *users_tweet_classifications* table represented in code 4 which has foreign keys to link the *user_id* to the *users.id* field and the *tweet_id* to the *tweets.id* while the constraint is set to ensure that *user_id* and *tweet_id* is only stored once.

Code 5 is the code representing the creation of the *hashtags* table, which has a particular *hashtag_value* which needs to be unique, as a hashtag on Twitter is a unique identifier it works well, similarly to *users* it has a field range of 30 characters, and again similarly it has a table *hashtag_tweet_classifications provided in code 6* to relate a particular *hashtag* to a particular *tweet* where the combination of *hashtag_id and tweet_id* requires uniqueness with foreign keys to relate the *hashtag* to a *tweet* through *hashtags.id to hashtag_tweet_classifications.hashtag_id* and then to relate to individual tweets through the *hashtag_tweet_classifications.tweet_id* to the *tweets.id* relation.

3.4 User Interaction Design

This section will provide the information about how users will interact with the system through the use of an Information Flow Diagram, which shows the exchange of information between system components (Fakhroutdinov K, 2018) which gives an indication of how the system will pass information from one entity to another at a high level, another method used to show the user interaction which is included in this section is the use of Use Case Diagrams, which provide an overview of the requirements of a system (UML 2 Use Case Diagrams: An Agile Introduction, 2018).

3.4.1 Information Flow Diagram

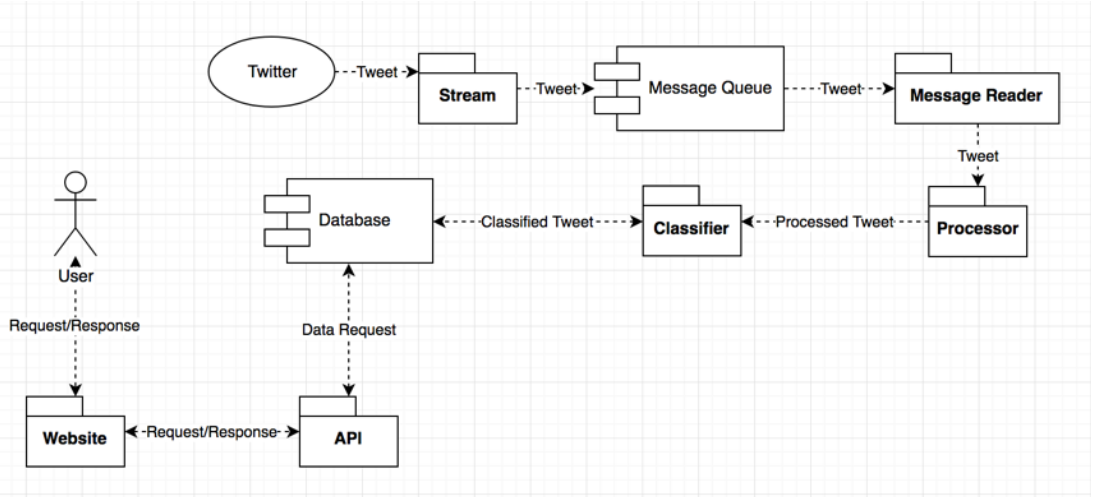


Figure 5 High-level Information flow diagram

Illustrated in figure 5 is the high-level information flow diagram for the project, where it shows the flow of data from the external service *Twitter* right down to the *Database* – and how the information is retrieved through data requests from the user. Starting at *Twitter*, the *Stream* retrieves an individual *Tweet* object which is then transferred on to the *Message Queue* – a *Message Reader* fetches the *Tweet* from this queue and dispatches the object to a *Processor* which transforms the original *Tweet* object in to a *Process Tweet* object which is then dispatched over to the *Classifier* which performs the classification task and creates a *Classified Tweet* object which is then stored in the *Database*. As referred to previously, this is the offline part which continuously works through incoming Tweets from Twitter, while on the left, the *User* makes requests to the website by navigating/searching sections which then makes a request call to the *API* which handles the request and fetches the data from the *Database*.

3.4.2 Use Case Diagrams

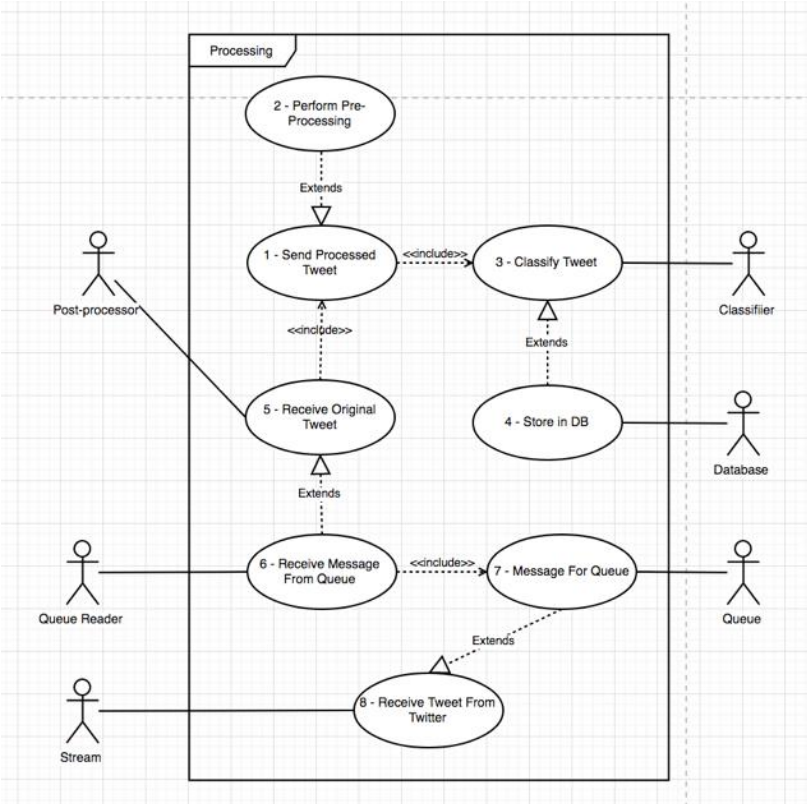


Figure 6 Use case diagram of the processing part

A use case diagram is a visualisation of the set of actions which a system can perform in connection with users of the system (UML 2 Use Case Diagrams: An Agile Introduction. 2018) – users of the system are not necessarily end-users and can represent different entities in the system, such as the use case diagram provided in figure 6 where the users are different services of the project and it shows the relationships with different use cases (the actions).

With the use case diagram illustrated in figure 6, it shows the relationship between the entities in the offline part of the project, such as how the *Stream* relates to use case 8, receiving a Tweet from Twitter which is extended by the use case 7 – which stores it as a message to be stored in the *Queue*. As it represents the entities of the system with particular actions, it is a helpful way to cross check requirements and to ensure completeness and the numbers relate to events/use cases which allow for traceability between requirements and use cases, as demonstrated in Appendix A where the final set of requirements is provided using Volere templates and additional use case diagrams are provided.

3.4.3 Activity Flow Diagram

An activity diagram is a flowchart to represent the flow from one activity to another activity, and usually described as an operation of the system (tutorialspoint.com, 2018), to visualise how a new Tweet will be processed on the system, figure 7 shows the activity diagram of a new Tweet entering the *Stream* services, and the flow of it through the system till it reaches the *Database*.

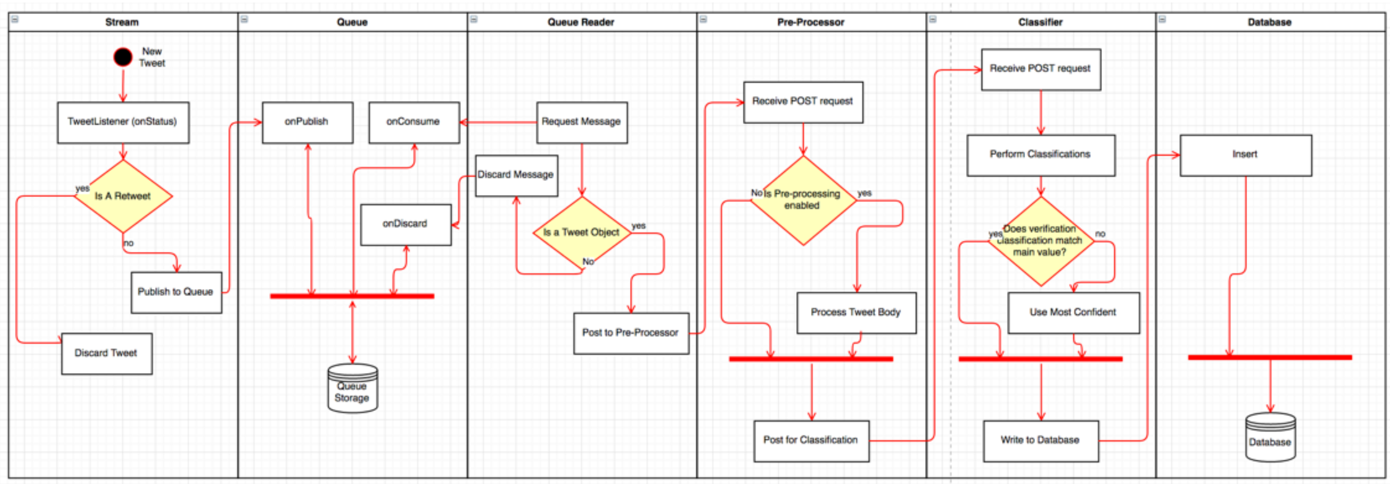


Figure 7 Activity diagram of a 'new tweet' entering the service

As illustrated in figure 7, a new Tweet will enter at the *Stream* which uses a Listener class to respond to new Tweets from the stream, from here it will check to see if the message is a Retweet, if it is – it'll discard it as a Retweet would mean processing a Tweet which has already been done, if it is not a Retweet then the message can be added to the queue, which is done through publishing it, the queue will handle the storage of the Tweet.

The *Queue Reader* runs independently and has a consumer which handles new messages from the *Queue*, the *Queue Reader* will ensure the format of the message is correct and then it posts it to the *Pre-Processor* if the format is correct, otherwise it discards it and removes it from the *Queue*. Once the *Pre-Processor* receives a POST request, it will check to see if *Pre-processing* is required, which is enabled by a configuration variable, if it is, then it will perform the pre-processing steps outlined in section 4 and then it will POST the processed Tweet for classification, if the pre-processing variable is not set then it will POST the Tweet without any processing done.

The *Classifier* receives the POST request and performs the classification using the two trained classifiers, and then does a check to see if the result is the same, if it is it can write the results to the *Database*, if the results differ then it will go based on a *tolerance weight* set up as a configuration variable, if the weight of the main classification is below the tolerance level, then it will use the verification classification result, otherwise if the weight is above the tolerance then it will use the main classification value. Using an activity diagram to visualise the flow of events helped in the development of the project as it helped to ensure tasks were being executed when expected.

3.5 Other Design Artefacts

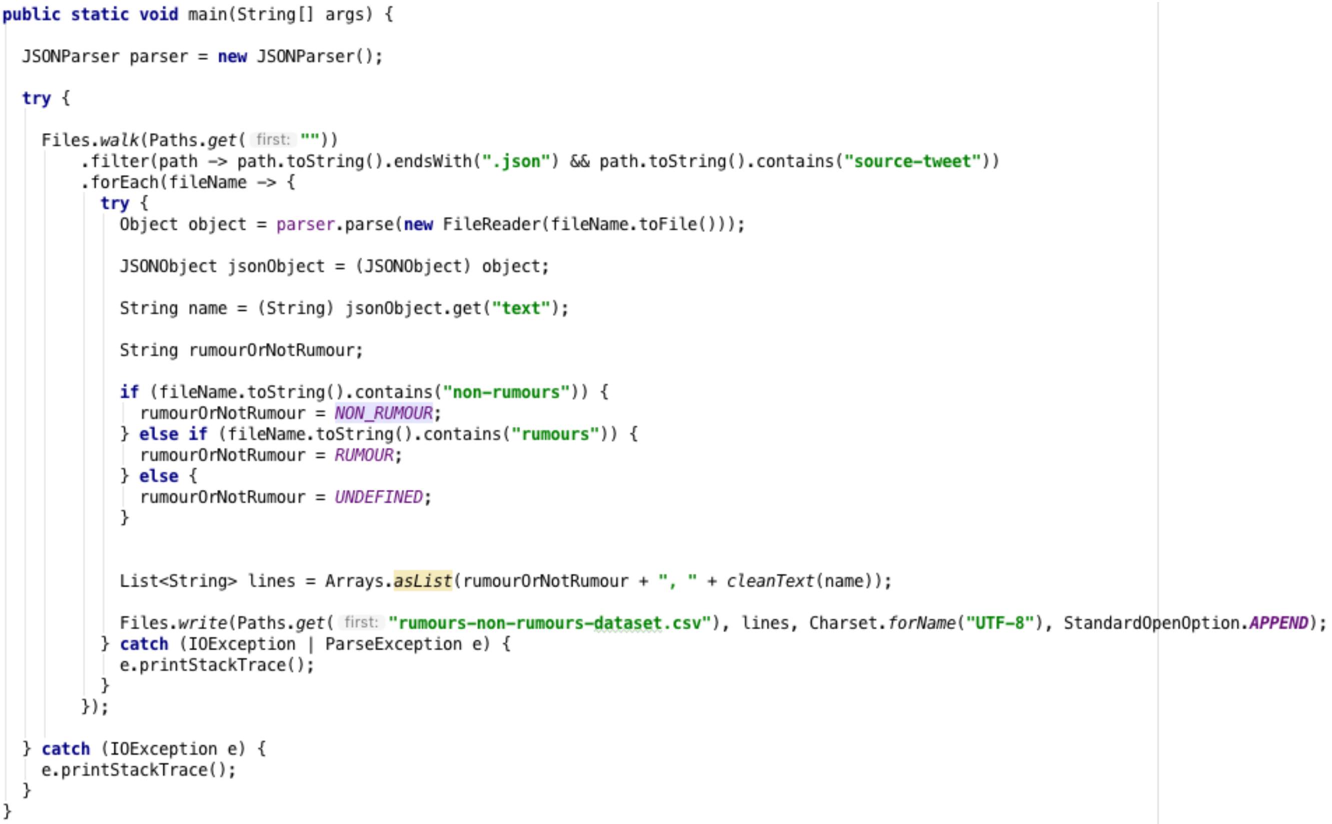
This section will outline the classification process which has been selected for the project, as the classification of a Tweet is the most important process a few considerations had to be made early to ensure that the project is as accurate as it can be.

3.5.1 Classification Process

As the project will analyse Tweets and classify them as either a rumour, or non-rumour this process is known as a binary classification, which is the process of classifying given document on the basis of a predefined class (Srivastava S and Kumari R, 2017). In order to classify a new document, in this case a Tweet as a rumour or non-rumour there are a few preliminary steps, first of all a dataset is required which can be used for training a classification model.

The project went with a dataset which contained a collection of Twitter rumours and non-rumours posted during 5 breaking news events (PHEME, 2018); which provided a dataset containing ~3800 non-rumours and ~2000 rumours, although swayed more towards non-rumours it was the most ideal for what was required as the dataset was a collection of Tweets, and the new documents would also be Tweets.

Some steps had to be performed on the dataset to allow it to be used with the classification models, as the original structure of the data was a folder per event, and within each of the five folders were a rumours and non-rumours folder which contained the Tweets as a JSON object[10], in order to use these Tweets, they had to be split in to a CSV format as follows; *label, text* where the *label* will either be rumour or non-rumour and the text will be the Tweet text of the Tweet object, code listing 7 shows the code which created the CSV file;



```java
public static void main(String[] args) {

  JSONParser parser = new JSONParser();

  try {

    Files.walk(Paths.get( first: ""))
        .filter(path -> path.toString().endsWith(".json") && path.toString().contains("source-tweet"))
        .forEach(fileName -> {
          try {
            Object object = parser.parse(new FileReader(fileName.toFile()));

            JSONObject jsonObject = (JSONObject) object;

            String name = (String) jsonObject.get("text");

            String rumourOrNotRumour;

            if (fileName.toString().contains("non-rumours")) {
              rumourOrNotRumour = NON_RUMOUR;
            } else if (fileName.toString().contains("rumours")) {
              rumourOrNotRumour = RUMOUR;
            } else {
              rumourOrNotRumour = UNDEFINED;
            }


            List<String> lines = Arrays.asList(rumourOrNotRumour + ", " + cleanText(name));

            Files.write(Paths.get( first: "rumours-non-rumours-dataset.csv"), lines, Charset.forName("UTF-8"), StandardOpenOption.APPEND);
          } catch (IOException | ParseException e) {
            e.printStackTrace();
          }
        });

  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

CODE 7 Java Code to convert the PHEME dataset in to a CSV file

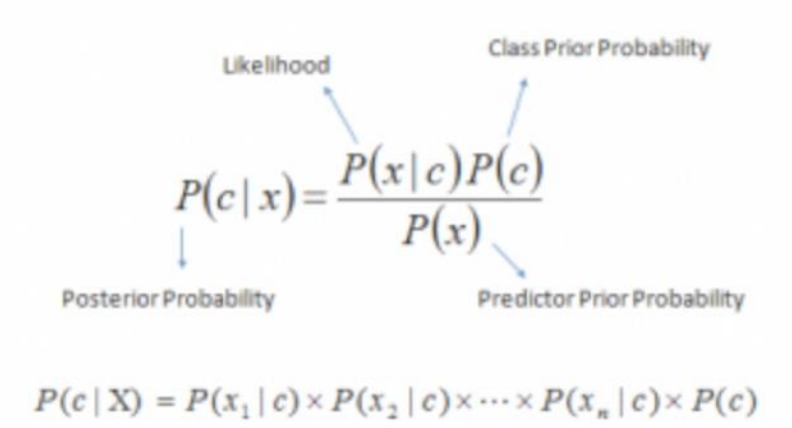
In code 7, it filters the files from (PHEME, 2018) dataset which end in '.json' and they are within a 'source-tweet' folder, the contents of which are then parsed as a JSON object, in order for the 'text' of the Tweet to be retrieved, as the JSON object contains other information, such as 'userId' etc. then based on if the file is in the rumour or non-rumour folder the label will be assigned, then this information is appended to a CSV file which is then used as the dataset to train the classification model.

As the classification will be done on natural language, a classification model had to be selected, and in the field of natural language classification there is often three classification models which get considered; Naïve Bayes, Maximum Entropy, and Support Vector Machines

[10] JavaScript Object Notation – a lightweight data-interchange format, which is easy for humans to understand and for machines to parse and generate (JSON, 2018)

(Go et al., 2009; Lee et al., 2011; Pang et al., 2002; Medlock B, 2008), early in the project plan it was swayed towards the use of Naïve Bayes as it was found to be drastically more efficient in terms of training time and individual classification time.

Naïve Bayes is a classification model based on Bayes Theorem and it assumes that the presences of a particular feature is unrelated to the presence of another feature – hence the term naïve (Analytics Vidhya, 2018). As a result, it can calculate the class through what is known in Bayes' Theorem as the posterior probability, illustrated in figure 8.



$$P(c \mid x) = \frac{P(x \mid c)P(c)}{P(x)}$$

$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Figure 8 Bayes' Theorem Posterior Probability equation (Analytics Vidhya, 2018)

Which can be summarised as follows; given a set of attributes (*X*) the probability of the class (*c*) – *P(c|X)* where *X* is a set of attributes, in the case of natural language processing it would be text split in to a feature list, and *c* would be the class/label which is being evaluated is equal to the probability of $x_{1\ldots n}$ belonging to class *c* multiplied by the class prior probability (*P(c)*) which is the maximum likelihood of something belonging to that class a rough example is as follows;

$$X\colon \begin{cases} this \\ sentence \\ requires \\ classification \end{cases}$$

$$\begin{aligned} P('rumour'|X) = {} & P('this' \mid 'rumour') * P('sentence'|'rumour') * P('requires'|'rumour') \\ & * P('classification'|'rumour') * P('rumour) \end{aligned}$$

The same equation would be done for 'non-rumour' and the class the highest posterior probability value would be selected as the appropriate label for the text.

4. System Implementation

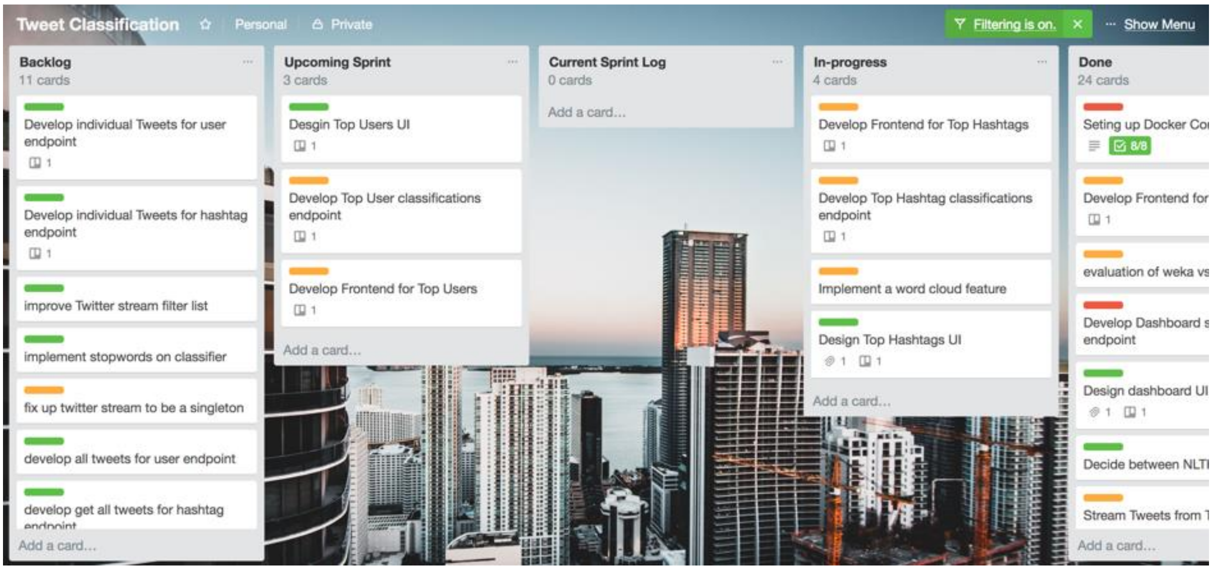
This section will outline the implementation of the project and how it was achieved, discussing the rationale for the tools, languages and other aspects which have been selected as well as providing evidence on the amount of work produced and the use of version control.

4.1 Summary and Rationale for tools, languages, databases, APIs and frameworks

The project aimed to utilise the best tools, languages, and frameworks available at the time of starting the project, and the decisions were made to reflect the knowledge and ability of the developer; some of the tools selected was to ease the development and speed up the progress.

4.1.1 *Tools*

The tools selected for the project were mainly for the management side of things and making sure the project was kept on track, and tools which were selected to aid with development, one of the core tools which was vital for the delivery of the project was *Trello*[11].



Tweet Classification

Backlog
11 cards
- Develop individual Tweets for user endpoint
- Develop individual Tweets for hashtag endpoint
- improve Twitter stream filter list
- implement stopwords on classifier
- fix up twitter stream to be a singleton
- develop all tweets for user endpoint
- develop get all tweets for hashtag endpoint

Add a card...

Upcoming Sprint
3 cards
- Desgin Top Users UI
- Develop Top User classifications endpoint
- Develop Frontend for Top Users

Add a card...

Current Sprint Log
0 cards

Add a card...

In-progress
4 cards
- Develop Frontend for Top Hashtags
- Develop Top Hashtag classifications endpoint
- Implement a word cloud feature
- Design Top Hashtags UI

Add a card...

Done
24 cards
- Seting up Docker Co
- Develop Frontend for
- evaluation of weka vs
- Develop Dashboard s endpoint
- Design dashboard UI
- Decide between NLTI
- Stream Tweets from T

Add a card...

Figure 9 Trello board for project

Figure 9 illustrates the *Trello* board which was used to keep track of which stories[12] were currently in progress and what still needed done; a good visualisation tool to record progress and as it is online it means it can be accessed from anywhere, rather than using a physical board.

For the development side of things, as a student the developer had access to all the Jetbrains[13] tools which could aid with the development, for the project the use of *IntelliJ Ultimate* was used, as it is an IDE to be used with the Java Programming Language and something which the developer is familiar in using, another Jetbrains tool which had been selected is *Datagrip* which is an IDE designed for managing databases and writing SQL which is important as the project will utilise the use of a SQL Database.

For maintaining and deploying of the project, the use of *Docker* was selected, as it is described as an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data centre VMs, or the cloud (Docker, 2018) – with *Docker* the project could be built with the confidence knowing that it could run anywhere.

For building the project and managing the dependencies the use of *Gradle* was used, which is build tool to help build, automate and deliver better software (Gradle, 2018). *Gradle* meant that the developer did not have to worry about dependencies, and the version of packages

[11] Trello is the easy, free, flexible, and visual way to manage your projects and organize anything, trusted by millions of people from all over the world. (About | What is Trello?, 2018)

[12] A user story is a tool used in Agile software development to capture a description of a software feature from an end-user perspective. (SearchSoftwareQuality, 2018)

[13] https://www.jetbrains.com/

which were needed, as *Gradle* does this for you. As seen in section 4.1.2, one of the core development languages is *Java* and with *Java* if dependencies are not managed correctly then it can lead to errors at runtime etc. which is why the use of a build tool is recommended, and *Gradle* has been selected as the developer is familiar with it.

Part of the system as witnessed in figure 1 required the use of a *Queue*, the use of *RabbitMQ* was selected as it is lightweight and easy to deploy and manage (RabbitMQ, 2018) and as it has support for *Java* through its API it was ideal as it fitted in nicely to the rest of the system architecture.

4.1.2 Languages

The project made use of languages which the developer was most confident in using, as the project was primarily a collection of web applications, the use of *Java EE* was utilised as it allows for the development of scalable web applications (Java EE, 2018), all the code was built using the build tool discussed in section 4.1.1 as runnable *WAR* files[14] which were deployed on *Tomcat* servlets which are running within *Docker* containers as discussed in section 3.1.

Java EE is built on top of the *Java SE* (Differences between Java EE and Java SE, 2018) describe how *Java SE* provides the core functionality of the Java programming language, such as the basic types, objects and classes that allow for the creation of a Java application, while *Java EE* extends on this by providing an API for developing and running networked applications.

For the development of the frontend, HTML, CSS and some JavaScript have been selected, as these are core languages to be used in frontend development, the Java web application will pass the required variables and data to the HTML pages using a templating engine which parses the data and renders it correctly, a template engine allows pre-built HTML pages that allows for the information to be plug in at rendering time; the use of Handlebars[15] was selected as the templating engine as it is suited for Java and based on the popular Mustache[16] templating engine, meaning it is well documented.

4.1.3 Database

As discussed in section 3.3.2 the project went with a SQL database design, when it comes to databases which support SQL there are various types, *PostgreSQL, Oracle SQL, SQL Server, SQLite* amongst others, and choosing one usually comes down to preference although it is good to consider the flexibility, performance and scalability (Lifewire, 2018) if the database is going be running server side, which the project will utilise.

The project went with *MySQL* as its database choice, as it is considered one of the most popular open-source databases which offers high-performance and scalability (MySQL, 2018) and it a database technology which the developer is most use too meaning it was the most suited out the possibilities that were available.

4.1.4 APIs

[14] WAR files are used to combine JSPs, servlets, Java class files, XML files, javascript libraries, JAR libraries, static web pages, and any other resources needed to run the application. (Understanding WAR, 2018)

[15] https://github.com/jknack/handlebars.java

[16] http://mustache.github.io/mustache.5.html

An Application Programming Interface (API), is described as a software intermediary that allows two applications to talk to each other (MuleSoft, 2018) and specifies how software components should interact by providing the building blocks, and the developer needs to put them together (Webopedia, 2018).

In terms of this project it required the use of an external API which is the *Twitter Filter Real-time Tweets API* (POST statuses/filter — Twitter Developers, 2018) and it was compulsory for retrieving a life feed of Tweets in to the service. In order to communicate with this API the use of a library known as *Twitter4J* was used which as mentioned in section 3.1 is known as the unofficial Java Library for the Twitter API (Twitter4j, 2018) and provides a collection of methods for interacting with the Twitter API which sped up implementation time as it contained all the required classes for retrieving live Tweets.

The project itself made use of internal APIs for communication between its services, such as the *Processing* service could communicate to the *Classifier* service through an API which was developed, this relied on the use of *Jersey* which is a Java framework for creating RESTful services which allow communication through various resource types (Jersey, 2018), with the use of this framework it meant that each of the services could communicate to each other by passing JSON objects, which would then be handled by the appropriate service, this helped to decouple the code, and meant that if one service needed to change, it could be done without affecting the other service, providing the *Interface*[17] is kept the same.

4.1.5 Frameworks and Libraries

The project made use of Bootstrap framework/library, as it is designed to make the development of a website faster and easier by providing high quality JavaScript and CSS to build a more responsive site (Bootstrap, 2018), and as it is considered to be easy to use and provides responsive features (Bootstrap Get Started, 2018) it was an ideal choice for the developer, as they had little experience in this area.

As briefly mentioned in section 4.1.4, the *Jersey* framework was used in order to help with the development of the project as it extends on the *Java JAX-RS*[18] implementation to aid in the development of a *RestFUL* service (Jersey, 2018), and had been selected as it is considered lightweight and worked nicely with the rest of the system architecture.

As most of the project dealt with exchanging data through JSON, a Java Library for easy parsing of JSON objects in to Java objects and vice-versa was required, which is why the use of Jackson JSON[19] was selected as it allows for efficient serialisation of a Java object in to a JSON object and vice-versa (tutorialspoint.com, 2018) and works nicely with Jersey as it is built with *JAX-RS* in-mind.

For the frontend, JQuery is also in use as it is a nice library which speeds up interacting with the website easier as it allows a clean API for Document Object Model (DOM) manipulation, animation and AJAX (Asynchronous JavaScript and XML) (JQuery Foundation, 2018), the use of JQuery allows for the interactivity of the website, and only displays content which is required allowing for smoother performance, another frontend framework/library which had

[17] A device or program for connecting two items of hardware or software so that they can be operated jointly or communicate with each other. (Oxford Dictionaries, 2018)
[18] Java API for RESTful Web Services
[19] https://github.com/FasterXML/jackson-docs

been selected was Chart.JS[20], as it is a simple JavaScript framework for creating interactive charts, and supports a range of chart types.

For performing the classification there are lots of libraries available, there were two main ones which had been considered as they are popular classification libraries for Java, and as Java was the language of choice, both had to be considered as they are equally popular, and as previously mentioned in section 3.1, both Weka and Mallet were evaluated and provided similar results, and neither one was "better" than the other.

It was determined to do a two-stage classification process, where the use of Mallet was used to perform a classification and get a label and weight value, then the same instance went through classification with the Weka library, and then the labels were compared – if they are equivalent then this was the label that will get assigned, otherwise the weight from Mallet will be evaluated and if it is below a configurable value then it will go with the classification label from Weka – although both libraries used the same classification algorithm, as both were Naïve Bayes classifiers, the features from each library differ and there will be variation between results, which is why the choice of using both was made.

4.2 Use of Version Control

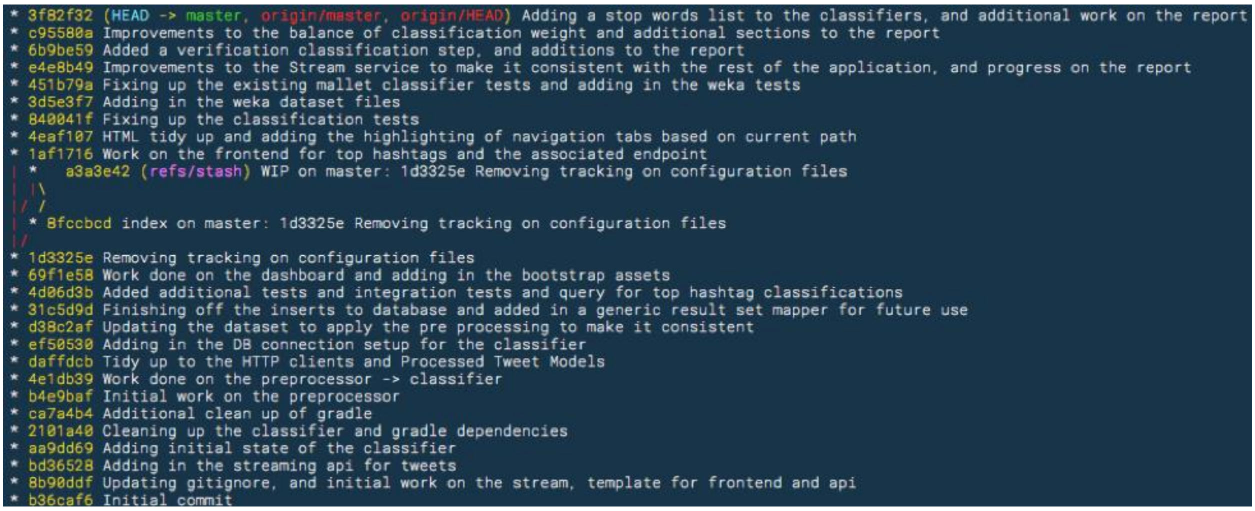
Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later (Git – About Version Control, 2018) and is a way to keep track of particular versions or see changes of a file, and is particularly useful in software as it means multiple people can work on the same file and conflicts can be controlled easier – but most importantly if a mistake is made the changes can be easily reverted, rather than having to manually make all the changes.

Git was the version control of choice for the project, as the developer has prior experience in using it, and it offers command line integration along with a GUI version to easily manage and maintain the versioning of the project.

Although a particular branching strategy is best used when working with version control if there are multiple developers, such as the common *Git flow*[21] branching strategy, as the project consisted of one developer no particular strategy was used other than applying a rule that only "complete" commits were allowed to be committed, i.e. if a commit contained code which didn't run then it was not allowed to be committed to the master branch – although it could still be stashed or stored in a separate branch.

[20] https://www.chartjs.org/

[21] http://nvie.com/posts/a-successful-git-branching-model/



```
* 3f82f32 (HEAD -> master, origin/master, origin/HEAD) Adding a stop words list to the classifiers, and additional work on the report
* c95580a Improvements to the balance of classification weight and additional sections to the report
* 6b9be59 Added a verification classification step, and additions to the report
* e4e8b49 Improvements to the Stream service to make it consistent with the rest of the application, and progress on the report
* 451b79a Fixing up the existing mallet classifier tests and adding in the weka tests
* 3d5e3f7 Adding in the weka dataset files
* 840041f Fixing up the classification tests
* 4eaf107 HTML tidy up and adding the highlighting of navigation tabs based on current path
* 1af1716 Work on the frontend for top hashtags and the associated endpoint
| *   a3a3e42 (refs/stash) WIP on master: 1d3325e Removing tracking on configuration files
| |\
|/ /
| * 8fccbcd index on master: 1d3325e Removing tracking on configuration files
|/
* 1d3325e Removing tracking on configuration files
* 69f1e58 Work done on the dashboard and adding in the bootstrap assets
* 4d06d3b Added additional tests and integration tests and query for top hashtag classifications
* 31c5d9d Finishing off the inserts to database and added in a generic result set mapper for future use
* d38c2af Updating the dataset to apply the pre processing to make it consistent
* ef50530 Adding in the DB connection setup for the classifier
* daffdcb Tidy up to the HTTP clients and Processed Tweet Models
* 4e1db39 Work done on the preprocessor -> classifier
* b4e9baf Initial work on the preprocessor
* ca7a4b4 Additional clean up of gradle
* 2101a40 Cleaning up the classifier and gradle dependencies
* aa9dd69 Adding initial state of the classifier
* bd36528 Adding in the streaming api for tweets
* 8b90ddf Updating gitignore, and initial work on the stream, template for frontend and api
* b36caf6 Initial commit
```

Figure 10 Git log output for 'master' branch

Illustrated in figure 10 is the commits which have been made to the master branch, all of which are bits of work which if checked out would provide an indication of the progress and allow a working product at that particular time.

4.3 Summary of Volume of Code Produced

The project was split in to 6 services, along with a common package of classes/methods which was used in more than one service, table 5 provides the volume of code produced in each package, referred to as ‘service name’.

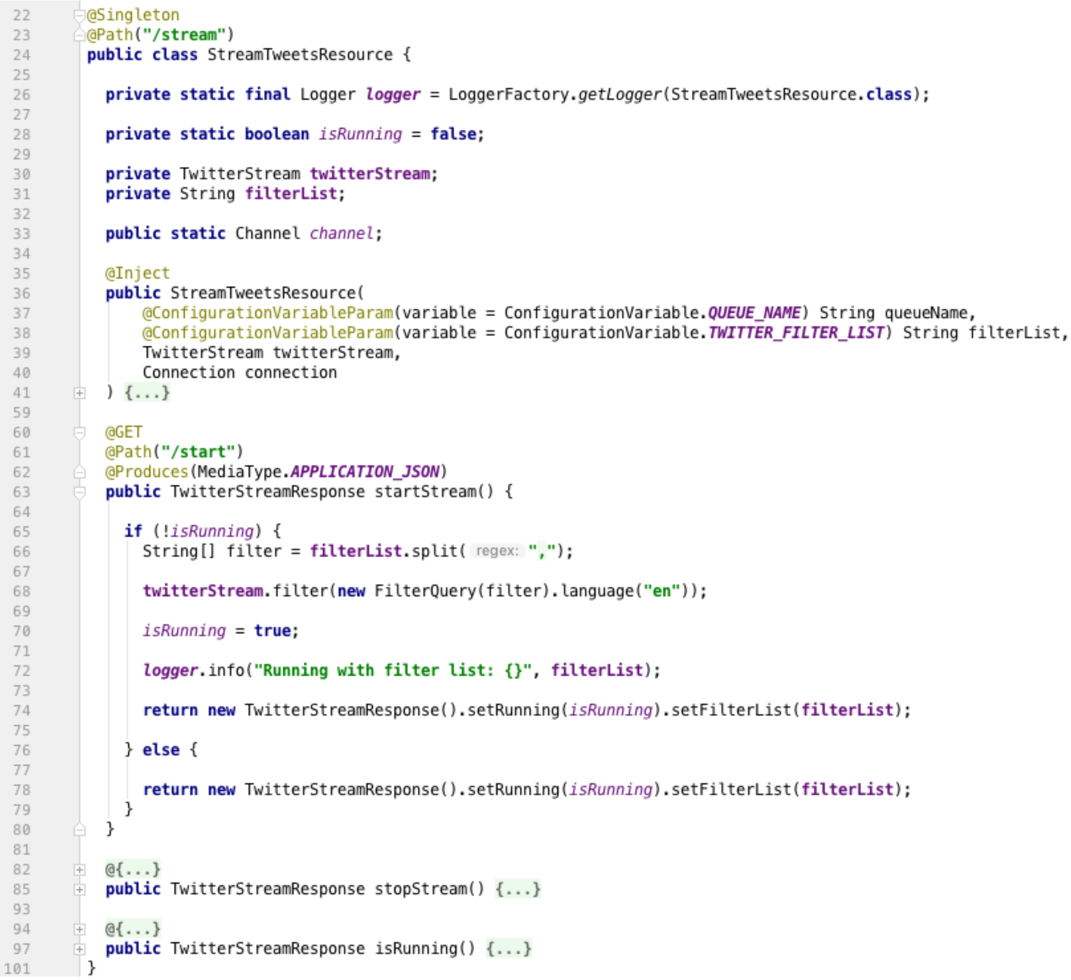
Table 5 Volume of Code produced per service

Service Name	Classes	Methods	SQL Queries	HTML Pages	JavaScript	CSS
API	21	30	15	0	0	0
Classifier	30	47	5	0	0	0
Common	34	60	0	0	0	0
Frontend	11	20	0	6	5	3
Pre-processor	4	6	0	0	0	0
Queue Reader	5	7	0	0	0	0
Stream	6	15	0	0	0	0
Total	111	185	20	6	5	3

4.4 System Walkthrough

The system walkthrough is simulated on the development system where the localhost is mapped to exposed ports running in the Docker containers, to help with the walkthrough it is assumed that the Stream has not been initiated. Beginning with the *Stream, it* is initiated by issuing a GET request to https://localhost:9001/stream/start which is mapped to port 8080 on the Docker container running the *Stream* which contains a WAR file which maps the requests to a class which extends a Jersey server configuration which binds the required classes that will be needed.

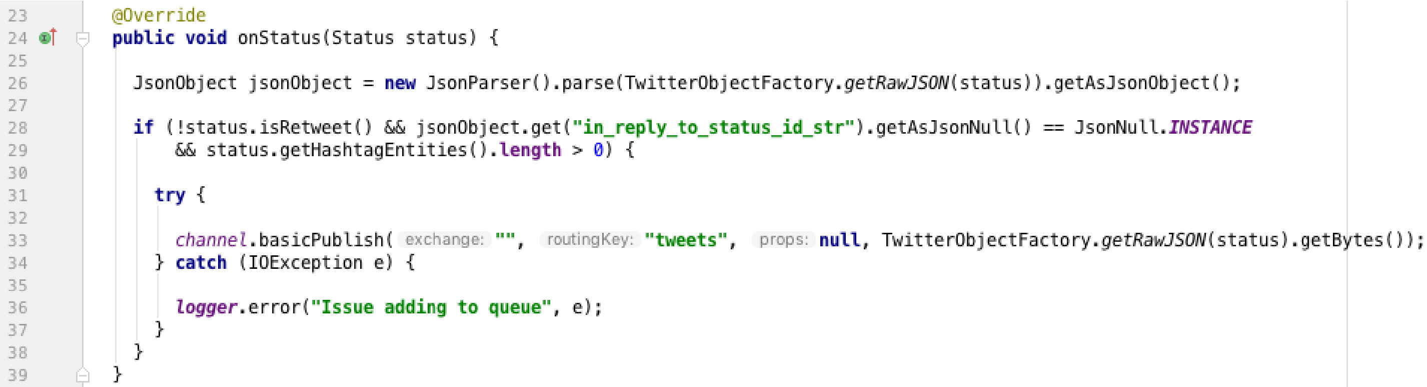
As Jersey is being used, it handles the GET request by finding the /stream/start path which is annotated in the StreamTweetsResource() as seen in code listing 8 where any requests to /stream will come to this class and any request to /start will go to the method *startStream()*.



```
22   @Singleton
23   @Path("/stream")
24   public class StreamTweetsResource {
25
26     private static final Logger logger = LoggerFactory.getLogger(StreamTweetsResource.class);
27
28     private static boolean isRunning = false;
29
30     private TwitterStream twitterStream;
31     private String filterList;
32
33     public static Channel channel;
34
35     @Inject
36     public StreamTweetsResource(
37         @ConfigurationVariableParam(variable = ConfigurationVariable.QUEUE_NAME) String queueName,
38         @ConfigurationVariableParam(variable = ConfigurationVariable.TWITTER_FILTER_LIST) String filterList,
39         TwitterStream twitterStream,
40         Connection connection
41     ) {...}
59
60     @GET
61     @Path("/start")
62     @Produces(MediaType.APPLICATION_JSON)
63     public TwitterStreamResponse startStream() {
64
65       if (!isRunning) {
66         String[] filter = filterList.split( regex: ",");
67
68         twitterStream.filter(new FilterQuery(filter).language("en"));
69
70         isRunning = true;
71
72         logger.info("Running with filter list: {}", filterList);
73
74         return new TwitterStreamResponse().setRunning(isRunning).setFilterList(filterList);
75
76       } else {
77
78         return new TwitterStreamResponse().setRunning(isRunning).setFilterList(filterList);
79       }
80     }
81
82     @{...}
85     public TwitterStreamResponse stopStream() {...}
93
94     @{...}
97     public TwitterStreamResponse isRunning() {...}
101  }
```

CODE 8 StreamTweetResource /stream/start

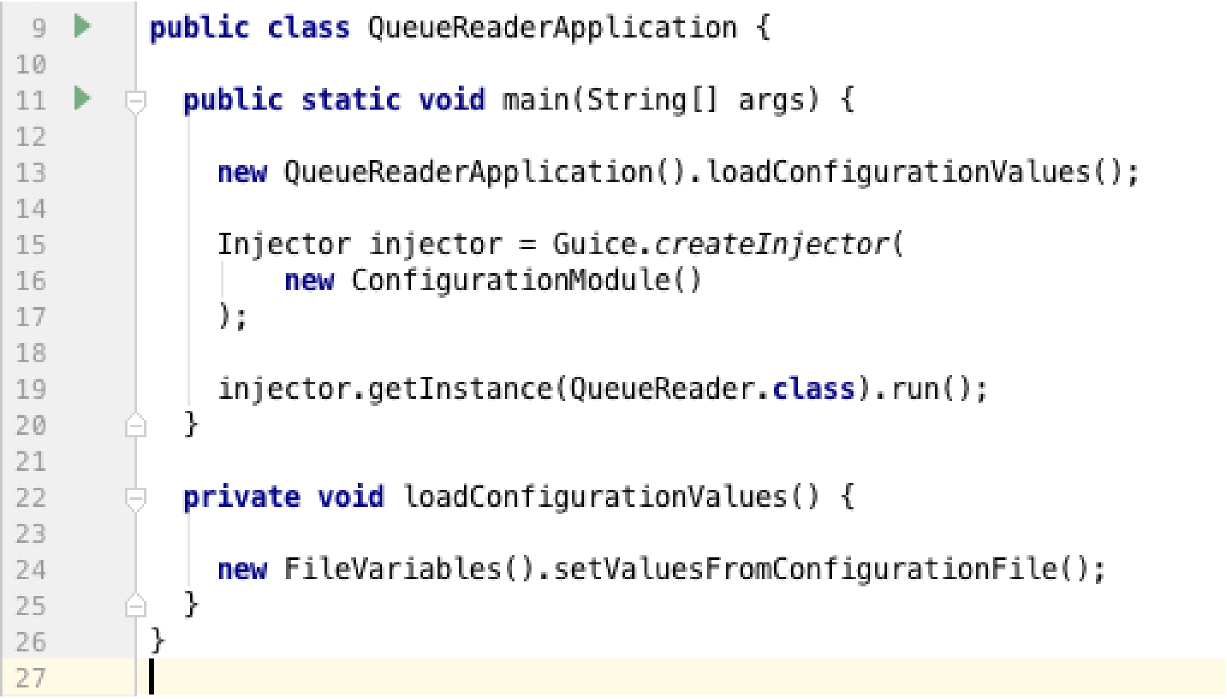
In code 8 on like 68, *twitterStream.filter(…)* is where the Twitter API Real-time filter stream is initialised, the *twitterStream* is a class from Twitter4J library which had been built and injected in to the class and a listener is attached to it, which has an *onStatus* method which is invoked when a new Tweet is streamed through the API, the code of which can be seen in code listing 9.



```
23   @Override
24   public void onStatus(Status status) {
25
26     JsonObject jsonObject = new JsonParser().parse(TwitterObjectFactory.getRawJSON(status)).getAsJsonObject();
27
28     if (!status.isRetweet() && jsonObject.get("in_reply_to_status_id_str").getAsJsonNull() == JsonNull.INSTANCE
29         && status.getHashtagEntities().length > 0) {
30
31       try {
32
33         channel.basicPublish( exchange: "",  routingKey: "tweets",  props: null, TwitterObjectFactory.getRawJSON(status).getBytes());
34       } catch (IOException e) {
35
36         logger.error("Issue adding to queue", e);
37       }
38     }
39   }
```

CODE 9 onStatus method which is invoked when a New Tweet is streamed in to the service

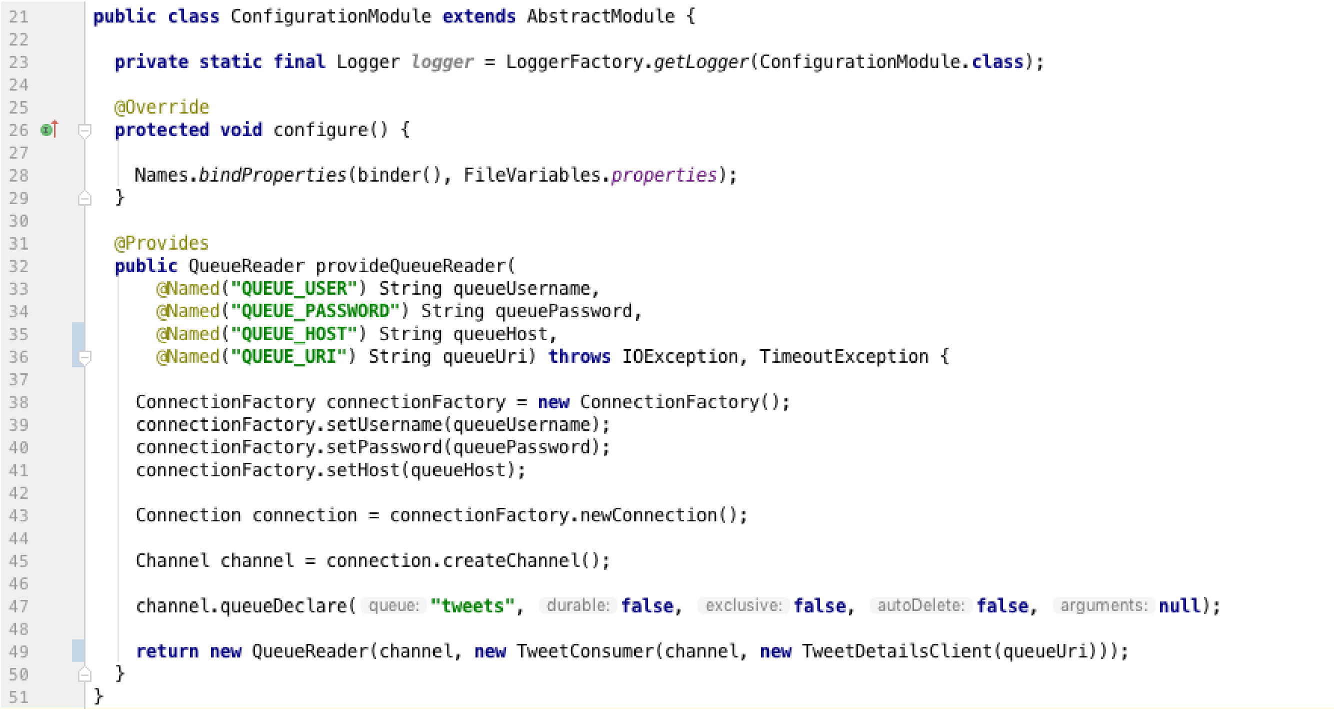
Once the new Tweets follow the condition that it is not a Retweet, and it contains some hashtags, then they get loaded on to a queue as seen on line 33 in code listing 9. Once the Tweets, which are converted to messages are on the *Queue* they are free to be consumed by the *Queue Reader*. The *Queue Reader* is a packaged JAR file that has a *main* method which is illustrated in code listing 10;



```
 9  public class QueueReaderApplication {
10
11      public static void main(String[] args) {
12
13          new QueueReaderApplication().loadConfigurationValues();
14
15          Injector injector = Guice.createInjector(
16                  new ConfigurationModule()
17          );
18
19          injector.getInstance(QueueReader.class).run();
20      }
21
22      private void loadConfigurationValues() {
23
24          new FileVariables().setValuesFromConfigurationFile();
25      }
26  }
27
```

CODE 10 QueueReader Application main method which initialises the reading from queue

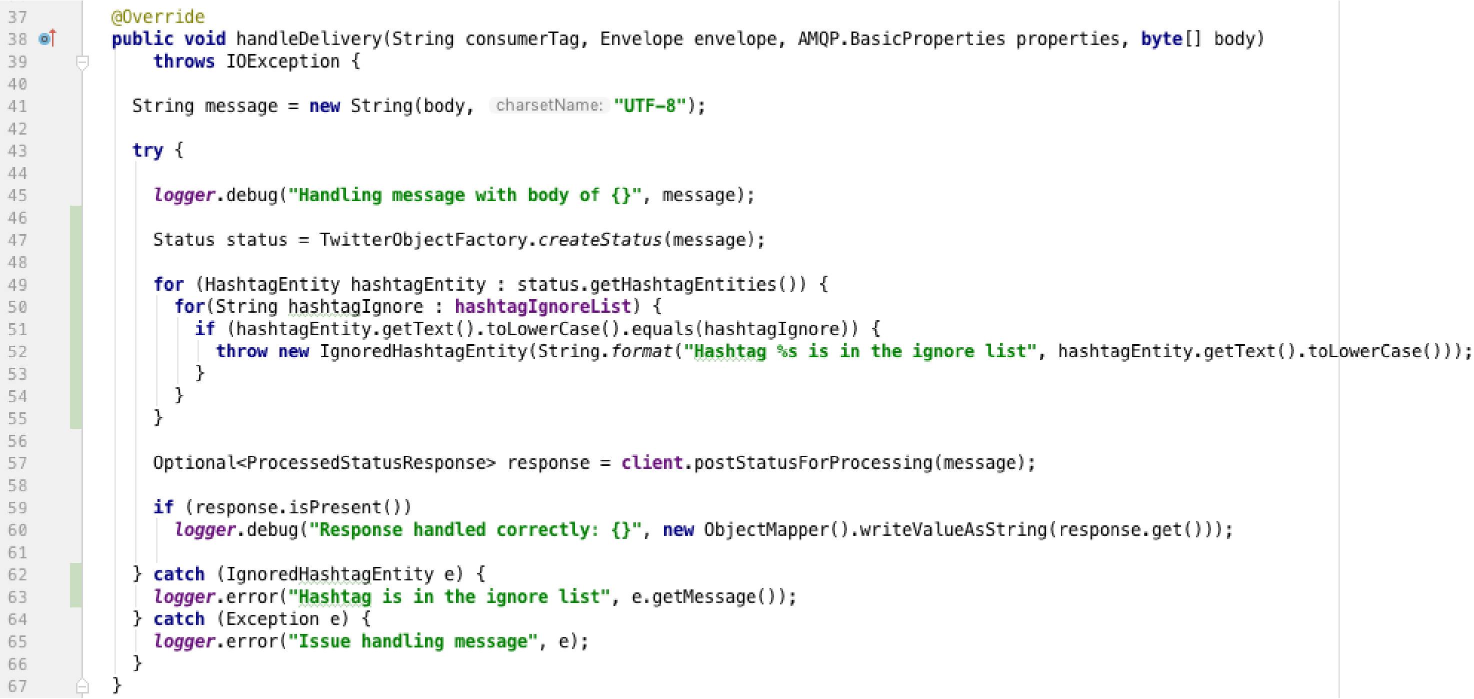
The *Queue Reader* makes use of the Google Guice library for dependency injection, as it makes it easier to inject services in to your code and makes the code easier to manage (GitHub – google/guice, 2018), on line 16 in code listing 10, the '*new ConfigurationModule()*' can be seen in code listing 11;



```
public class ConfigurationModule extends AbstractModule {

  private static final Logger logger = LoggerFactory.getLogger(ConfigurationModule.class);

  @Override
  protected void configure() {

    Names.bindProperties(binder(), FileVariables.properties);
  }

  @Provides
  public QueueReader provideQueueReader(
      @Named("QUEUE_USER") String queueUsername,
      @Named("QUEUE_PASSWORD") String queuePassword,
      @Named("QUEUE_HOST") String queueHost,
      @Named("QUEUE_URI") String queueUri) throws IOException, TimeoutException {

    ConnectionFactory connectionFactory = new ConnectionFactory();
    connectionFactory.setUsername(queueUsername);
    connectionFactory.setPassword(queuePassword);
    connectionFactory.setHost(queueHost);

    Connection connection = connectionFactory.newConnection();

    Channel channel = connection.createChannel();

    channel.queueDeclare( queue: "tweets",  durable: false,  exclusive: false,  autoDelete: false,  arguments: null);

    return new QueueReader(channel, new TweetConsumer(channel, new TweetDetailsClient(queueUri)));
  }
}
```

CODE 11 new ConfigurationModule code highlighing the power of Guice

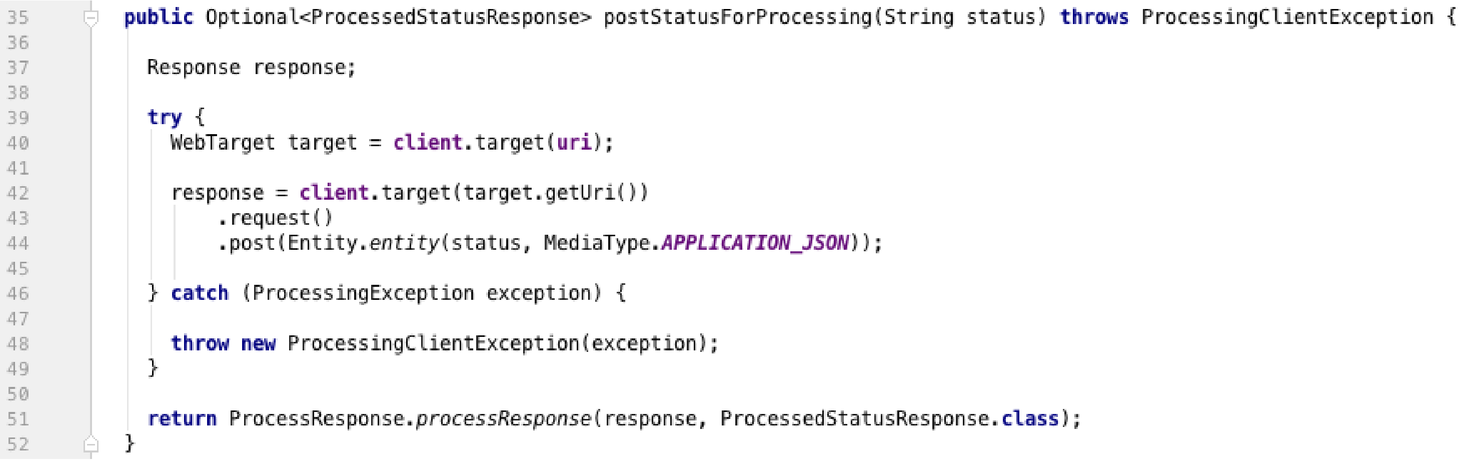
The *QueueReader* provided on line 25 is how Guice allows an easy facility to inject services, rather than having to create objects within the class, they can be separated out, making it easier to unit test at the testing stages. On line 49 from code listing 11 there is reference to a 'new TweetConsumer', this consumer is what handles the delivery from the *Queue* to the *Queue Reader* through the use of a overridden method in the *TweetConsumer* class which makes use of the *RabbitMQ* API which was mentioned in section 4.1.1, the consumer reads in the message from the *Queue* and then posts it for processing after doing a few checks, such as to make sure the Tweet doesn't contain any hashtags that are in the ignore list as witnessed in code listing 12;



```
37    @Override
38    public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties properties, byte[] body)
39        throws IOException {
40
41      String message = new String(body,  charsetName: "UTF-8");
42
43      try {
44
45        logger.debug("Handling message with body of {}", message);
46
47        Status status = TwitterObjectFactory.createStatus(message);
48
49        for (HashtagEntity hashtagEntity : status.getHashtagEntities()) {
50          for(String hashtagIgnore : hashtagIgnoreList) {
51            if (hashtagEntity.getText().toLowerCase().equals(hashtagIgnore)) {
52              throw new IgnoredHashtagEntity(String.format("Hashtag %s is in the ignore list", hashtagEntity.getText().toLowerCase()));
53            }
54          }
55        }
56
57        Optional<ProcessedStatusResponse> response = client.postStatusForProcessing(message);
58
59        if (response.isPresent())
60          logger.debug("Response handled correctly: {}", new ObjectMapper().writeValueAsString(response.get()));
61
62      } catch (IgnoredHashtagEntity e) {
63        logger.error("Hashtag is in the ignore list", e.getMessage());
64      } catch (Exception e) {
65        logger.error("Issue handling message", e);
66      }
67    }
```

CODE 12 Queue Reader handle delivery of message from the Queue

Once the *Queue Reader* has handled the Tweet it posts the message to the *Pre-processor*, as illustrated on line 57 in code listing 12, the *client.postStatusForProcessing* makes use of the *JAX-RS* functionality in *Java* to post the message to a specified URI, the code is provided in code listing 13;

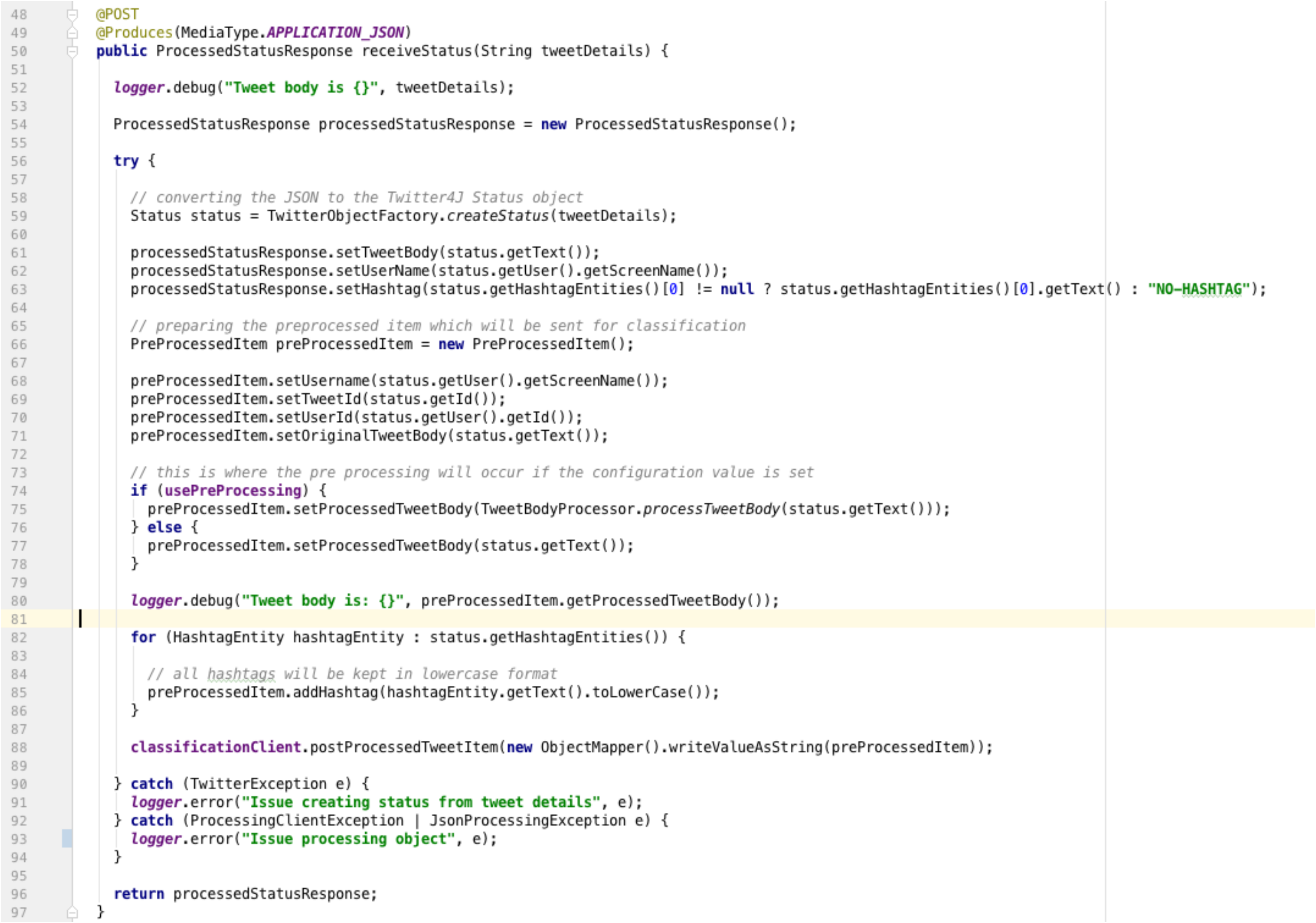


```
public Optional<ProcessedStatusResponse> postStatusForProcessing(String status) throws ProcessingClientException {

  Response response;

  try {
    WebTarget target = client.target(uri);

    response = client.target(target.getUri())
        .request()
        .post(Entity.entity(status, MediaType.APPLICATION_JSON));

  } catch (ProcessingException exception) {

    throw new ProcessingClientException(exception);
  }

  return ProcessResponse.processResponse(response, ProcessedStatusResponse.class);
}
```

CODE 13 Example of using JAX-RS client to post from one service to another

The *Pre-processor*, similar to other services makes use of *Jersey* which has a @POST path set as /process for the class *ReceiveTweetStatusResource*, once the *Queue Reader* executes the code in listing 13, the *Pre-processor* receives the Tweet message and can proceed with the processing, the

Pre-processing has one method which receives the POST message from the *Queue Reader* as provided in code listing 14;

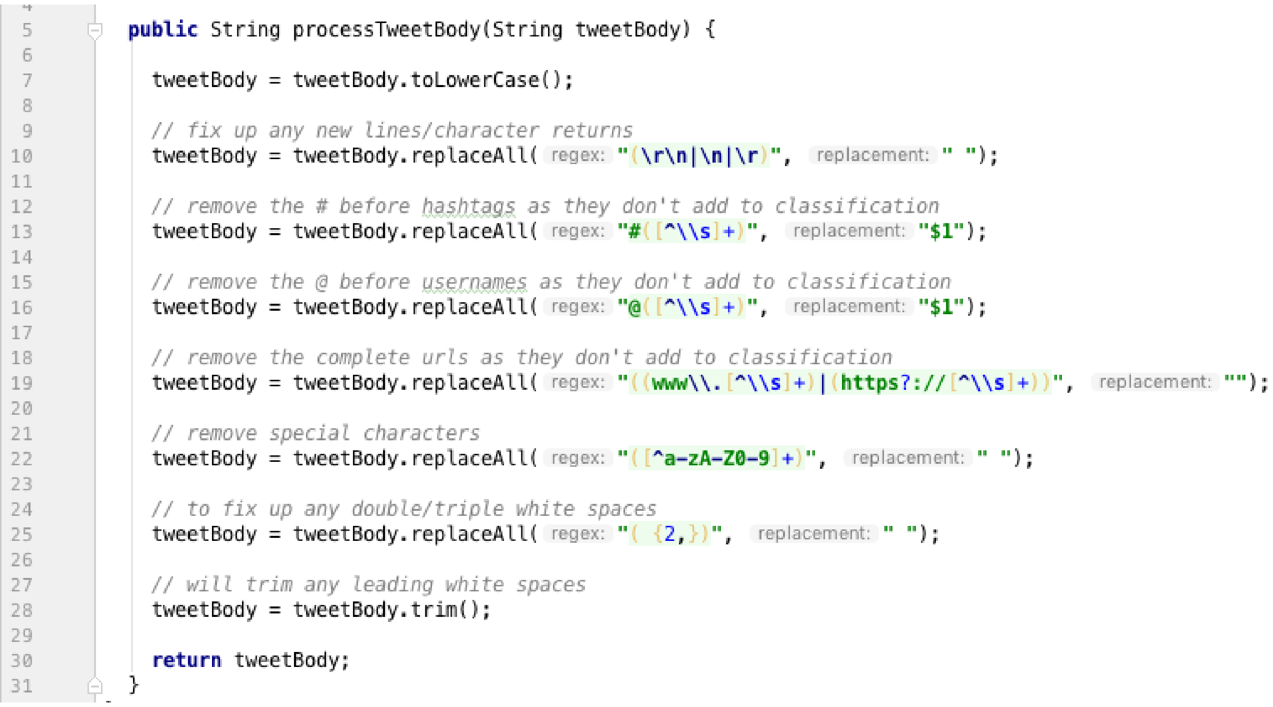


```
48  @POST
49  @Produces(MediaType.APPLICATION_JSON)
50  public ProcessedStatusResponse receiveStatus(String tweetDetails) {
51
52    logger.debug("Tweet body is {}", tweetDetails);
53
54    ProcessedStatusResponse processedStatusResponse = new ProcessedStatusResponse();
55
56    try {
57
58      // converting the JSON to the Twitter4J Status object
59      Status status = TwitterObjectFactory.createStatus(tweetDetails);
60
61      processedStatusResponse.setTweetBody(status.getText());
62      processedStatusResponse.setUserName(status.getUser().getScreenName());
63      processedStatusResponse.setHashtag(status.getHashtagEntities()[0] != null ? status.getHashtagEntities()[0].getText() : "NO-HASHTAG");
64
65      // preparing the preprocessed item which will be sent for classification
66      PreProcessedItem preProcessedItem = new PreProcessedItem();
67
68      preProcessedItem.setUsername(status.getUser().getScreenName());
69      preProcessedItem.setTweetId(status.getId());
70      preProcessedItem.setUserId(status.getUser().getId());
71      preProcessedItem.setOriginalTweetBody(status.getText());
72
73      // this is where the pre processing will occur if the configuration value is set
74      if (usePreProcessing) {
75        preProcessedItem.setProcessedTweetBody(TweetBodyProcessor.processTweetBody(status.getText()));
76      } else {
77        preProcessedItem.setProcessedTweetBody(status.getText());
78      }
79
80      logger.debug("Tweet body is: {}", preProcessedItem.getProcessedTweetBody());
81
82      for (HashtagEntity hashtagEntity : status.getHashtagEntities()) {
83
84        // all hashtags will be kept in lowercase format
85        preProcessedItem.addHashtag(hashtagEntity.getText().toLowerCase());
86      }
87
88      classificationClient.postProcessedTweetItem(new ObjectMapper().writeValueAsString(preProcessedItem));
89
90    } catch (TwitterException e) {
91      logger.error("Issue creating status from tweet details", e);
92    } catch (ProcessingClientException | JsonProcessingException e) {
93      logger.error("Issue processing object", e);
94    }
95
96    return processedStatusResponse;
97  }
```

CODE 14 Pre-processor POST method to receive messages from the Queue Reader

On line 74 in code listing 14, there is a check to see if *usePreProcessing* is true, this is to adhere to the requirements set for the project where it should be configurable to swap between the use of with or without processing, if the use of processing is set to true, which is provided as a configuration variable, it goes through the *processTweetBody* method on line 75 from code listing 14, which applies the following rules:

- Lower case the Tweet body, line 7 code listing 15
- Replace the # at the start of a hashtag, e.g. #doctor becomes doctor, line 13 code listing 15
- Replace the @ at the start of a user, e.g. @john_smith becomes john_smith, line 14 code listing 15
- Remove any URL, line 19 code listing 15
- To remove any double or more spaces, line 25 code listing 15
- To remove any new lines/character returns, line 10 code listing 15
- Remove any character which are not a letter or number, line 22 code listing 15
- And, to remove any trailing whitespaces, line 28 code listing 15



```java
public String processTweetBody(String tweetBody) {

  tweetBody = tweetBody.toLowerCase();

  // fix up any new lines/character returns
  tweetBody = tweetBody.replaceAll( regex: "(\r\n|\n|\r)",  replacement: " ");

  // remove the # before hashtags as they don't add to classification
  tweetBody = tweetBody.replaceAll( regex: "#([^\\s]+)",  replacement: "$1");

  // remove the @ before usernames as they don't add to classification
  tweetBody = tweetBody.replaceAll( regex: "@([^\\s]+)",  replacement: "$1");

  // remove the complete urls as they don't add to classification
  tweetBody = tweetBody.replaceAll( regex: "((www\\.[^\\s]+)|(https?://[^\\s]+))",  replacement: "");

  // remove special characters
  tweetBody = tweetBody.replaceAll( regex: "([^a-zA-Z0-9]+)",  replacement: " ");

  // to fix up any double/triple white spaces
  tweetBody = tweetBody.replaceAll( regex: "( {2,})",  replacement: " ");

  // will trim any leading white spaces
  tweetBody = tweetBody.trim();

  return tweetBody;
}
```

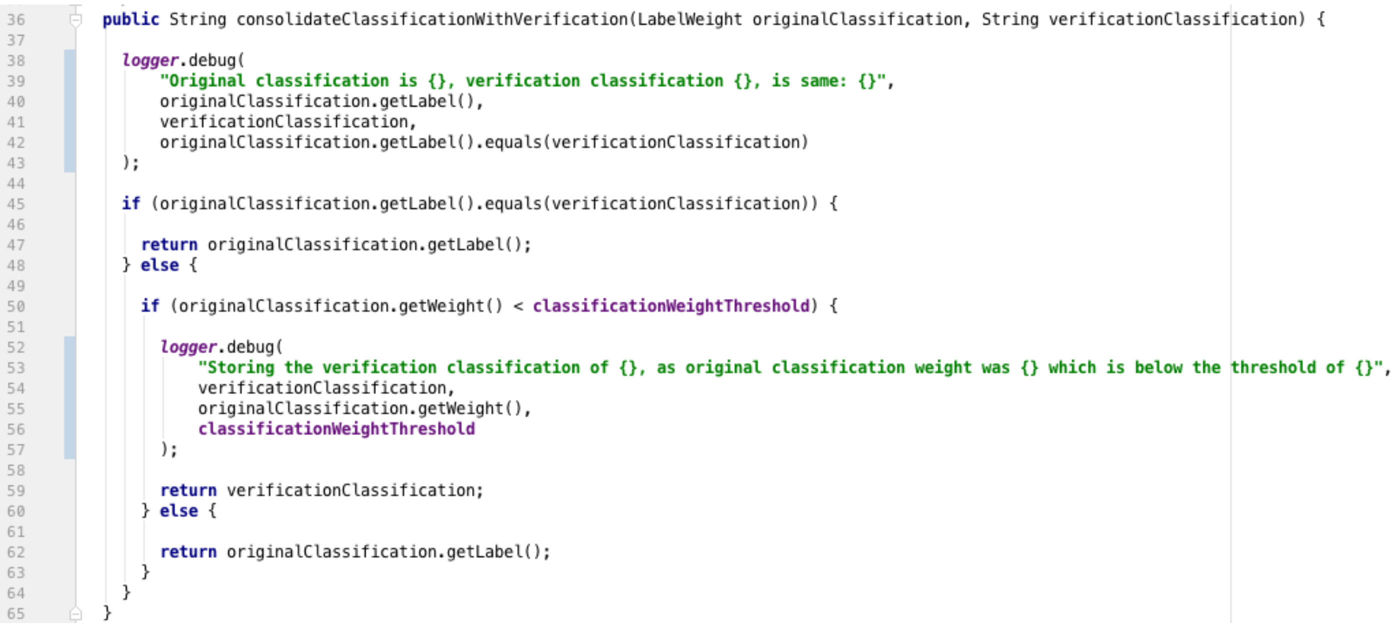
CODE 15 The processing rules which are applied when pre-processing is enabled

Once the Tweet has went through the processing steps, it gets posted to the *Classifier*, which is the last stage before storing to the *Database* as illustrated in figure 7 previously. Similarly, to the other services, the *Classifier* also is running with *Jersey* filters and has an annotated @POST method at the /classify path which the *Pre-processor* posts the processed message too, illustrated on line 88 in code listing 14.

The *Classifier* has a trained *Naïve Bayes* classification model using the Weka and Mallet libraries as a two-stage classification process was used as described previously in section 4.1.5, which follows the basic rules of;

- If the classification value match, then return that value
- If the classification values do not match, then check the Mallets class weight, if it is lower than the allowed limit return the Weka classification value
- If the classification weight is above the limit, then return Mallets class value

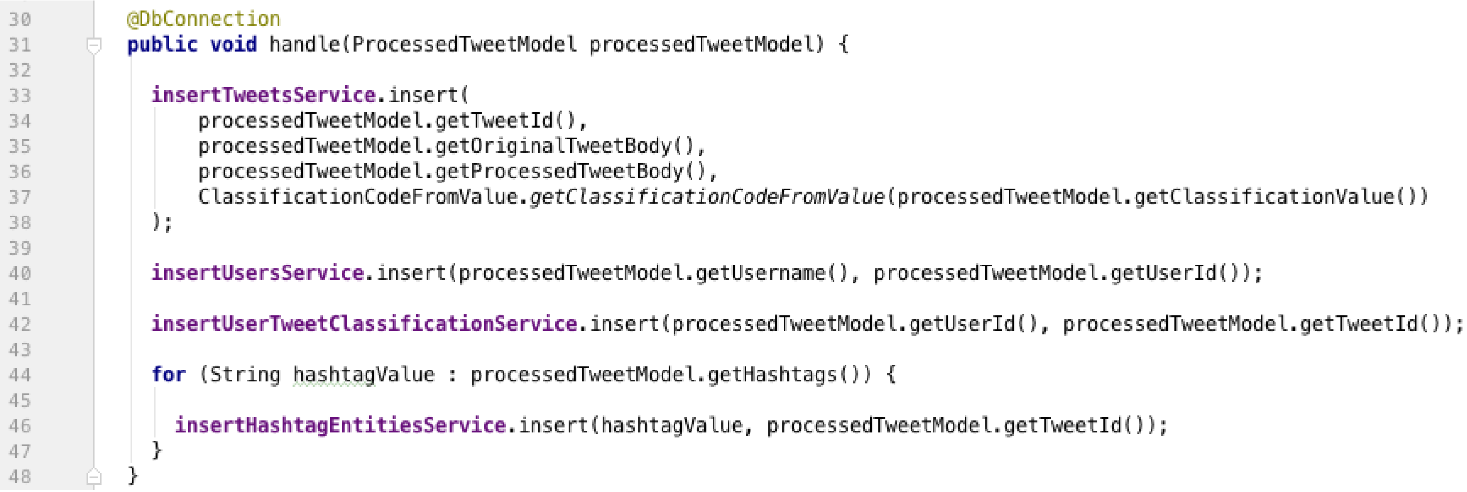
The code can be witnessed in the *consolidateClassificationWithVerification* method outlined in code listing 16;



```java
public String consolidateClassificationWithVerification(LabelWeight originalClassification, String verificationClassification) {

  logger.debug(
      "Original classification is {}, verification classification {}, is same: {}",
      originalClassification.getLabel(),
      verificationClassification,
      originalClassification.getLabel().equals(verificationClassification)
  );

  if (originalClassification.getLabel().equals(verificationClassification)) {

    return originalClassification.getLabel();
  } else {

    if (originalClassification.getWeight() < classificationWeightThreshold) {

      logger.debug(
          "Storing the verification classification of {}, as original classification weight was {} which is below the threshold of {}",
          verificationClassification,
          originalClassification.getWeight(),
          classificationWeightThreshold
      );

      return verificationClassification;
    } else {

      return originalClassification.getLabel();
    }
  }
}
```

CODE 16 Consolidation between the original classification value with the verification classification value

Once a classification label has been assigned, the results can be stored to the *Database*, a *HandleProcessedTweetService* class achieves this through the series of inserts for the necessary tables, which is illustrates in code listing 17.



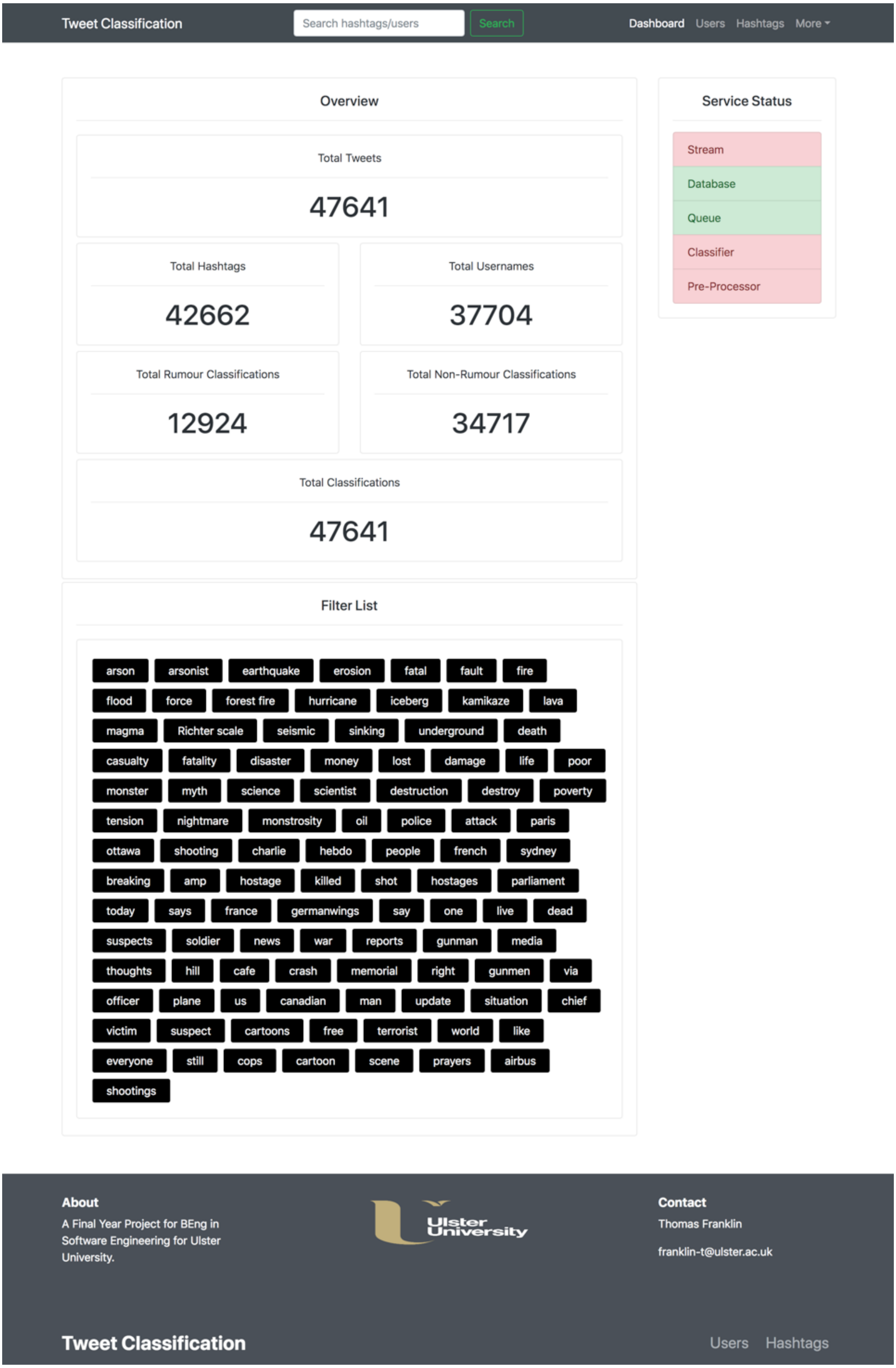
```java
30  @DbConnection
31  public void handle(ProcessedTweetModel processedTweetModel) {
32
33    insertTweetsService.insert(
34        processedTweetModel.getTweetId(),
35        processedTweetModel.getOriginalTweetBody(),
36        processedTweetModel.getProcessedTweetBody(),
37        ClassificationCodeFromValue.getClassificationCodeFromValue(processedTweetModel.getClassificationValue())
38    );
39
40    insertUsersService.insert(processedTweetModel.getUsername(), processedTweetModel.getUserId());
41
42    insertUserTweetClassificationService.insert(processedTweetModel.getUserId(), processedTweetModel.getTweetId());
43
44    for (String hashtagValue : processedTweetModel.getHashtags()) {
45
46      insertHashtagEntitiesService.insert(hashtagValue, processedTweetModel.getTweetId());
47    }
48  }
```

CODE 17 HandleProcessTweetSerive handle method to insert the processed Tweet in to Database

The service needs to insert in to the Tweets table, line 33 of code listing 17 once this is done it can then insert the Twitter user information and the hashtag information. Once the information is stored in the *Database* it ends the “offline” part of the system walkthrough.

For the “online” part of the service, figures 11, 12 and 13 show the various pages of the service which were modelled around the wireframes from section 3.2.2. Figure 11 illustrates the homepage/dashboard that is displayed when the user navigates to the home page route on the frontend service, while figure 12 shows the “top users” page which is reachable by the route /users on the frontend and finally figure 13 illustrates the “top hashtags” page which is reachable by the route /hashtags.

The frontend, similarly to the others and as described in section 4.1.4 the use of *Jersey* allows for annotated classes to match route resources that the user accesses through requests, while the frontend Java service communicates with API, which hides the requests from the user on the frontend, and they only receive and see the data which is required.



Tweet Classification

Search hashtags/users

Search

Dashboard Users Hashtags More

Overview

Total Tweets

47641

Total Hashtags	Total Usernames
42662	37704

Total Rumour Classifications	Total Non-Rumour Classifications
12924	34717

Total Classifications

47641

Service Status

🟥 Stream
🟩 Database
🟩 Queue
🟥 Classifier
🟥 Pre-Processor

Filter List

arson arsonist earthquake erosion fatal fault fire
flood force forest fire hurricane iceberg kamikaze lava
magma Richter scale seismic sinking underground death
casualty fatality disaster money lost damage life poor
monster myth science scientist destruction destroy poverty
tension nightmare monstrosity oil police attack paris
ottawa shooting charlie hebdo people french sydney
breaking amp hostage killed shot hostages parliament
today says france germanwings say one live dead
suspects soldier news war reports gunman media
thoughts hill cafe crash memorial right gunmen via
officer plane us canadian man update situation chief
victim suspect cartoons free terrorist world like
everyone still cops cartoon scene prayers airbus
shootings

About

A Final Year Project for BEng in Software Engineering for Ulster University.

Ulster University

Contact

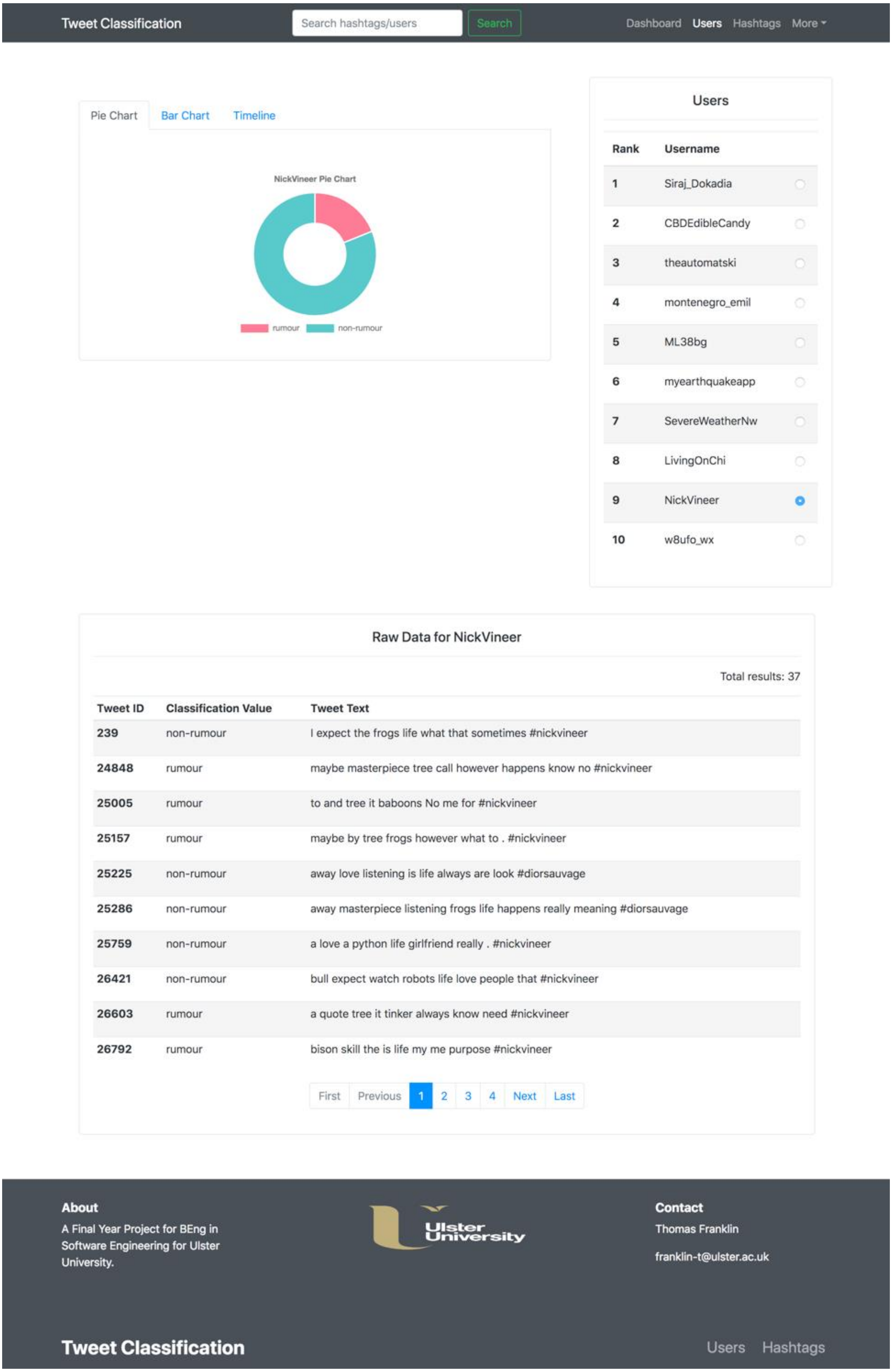
Thomas Franklin

franklin-t@ulster.ac.uk

Tweet Classification

Users Hashtags

Figure 11 Dashboard/Homepage webpage showing the various stats and status of the service



Users

Rank	Username	
1	Siraj_Dokadia	○
2	CBDEdibleCandy	○
3	theautomatski	○
4	montenegro_emil	○
5	ML38bg	○
6	myearthquakeapp	○
7	SevereWeatherNw	○
8	LivingOnChi	○
9	NickVineer	●
10	w8ufo_wx	○

Raw Data for NickVineer

Total results: 37

Tweet ID	Classification Value	Tweet Text
239	non-rumour	I expect the frogs life what that sometimes #nickvineer
24848	rumour	maybe masterpiece tree call however happens know no #nickvineer
25005	rumour	to and tree it baboons No me for #nickvineer
25157	rumour	maybe by tree frogs however what to . #nickvineer
25225	non-rumour	away love listening is life always are look #diorsauvage
25286	non-rumour	away masterpiece listening frogs life happens really meaning #diorsauvage
25759	non-rumour	a love a python life girlfriend really . #nickvineer
26421	non-rumour	bull expect watch robots life love people that #nickvineer
26603	rumour	a quote tree it tinker always know need #nickvineer
26792	rumour	bison skill the is life my me purpose #nickvineer

About

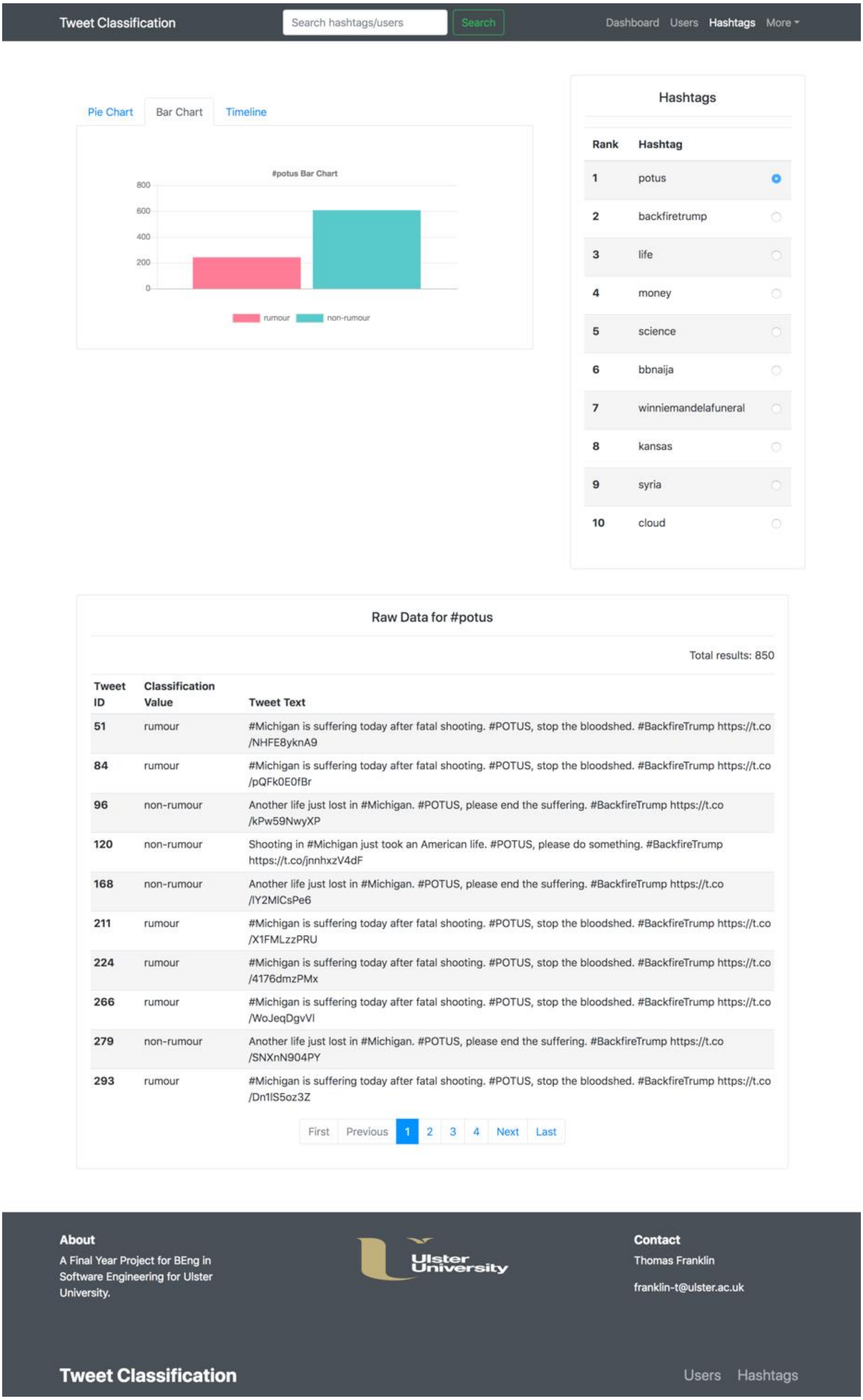
A Final Year Project for BEng in Software Engineering for Ulster University.

Contact

Thomas Franklin

franklin-t@ulster.ac.uk

Figure 12 /users page illustrating the use of bar charts to visualise the rumours/non-rumours



Hashtags

Rank	Hashtag	
1	potus	●
2	backfiretrump	○
3	life	○
4	money	○
5	science	○
6	bbnaija	○
7	winniemandelafuneral	○
8	kansas	○
9	syria	○
10	cloud	○

Raw Data for #potus

Total results: 850

Tweet ID	Classification Value	Tweet Text
51	rumour	#Michigan is suffering today after fatal shooting. #POTUS, stop the bloodshed. #BackfireTrump https://t.co/NHFE8yknA9
84	rumour	#Michigan is suffering today after fatal shooting. #POTUS, stop the bloodshed. #BackfireTrump https://t.co/pQFk0E0fBr
96	non-rumour	Another life just lost in #Michigan. #POTUS, please end the suffering. #BackfireTrump https://t.co/kPw59NwyXP
120	non-rumour	Shooting in #Michigan just took an American life. #POTUS, please do something. #BackfireTrump https://t.co/jnnhxzV4dF
168	non-rumour	Another life just lost in #Michigan. #POTUS, please end the suffering. #BackfireTrump https://t.co/lY2MlCsPe6
211	rumour	#Michigan is suffering today after fatal shooting. #POTUS, stop the bloodshed. #BackfireTrump https://t.co/X1FMLzzPRU
224	rumour	#Michigan is suffering today after fatal shooting. #POTUS, stop the bloodshed. #BackfireTrump https://t.co/4176dmzPMx
266	rumour	#Michigan is suffering today after fatal shooting. #POTUS, stop the bloodshed. #BackfireTrump https://t.co/WoJeqDgvVl
279	non-rumour	Another life just lost in #Michigan. #POTUS, please end the suffering. #BackfireTrump https://t.co/SNXnN904PY
293	rumour	#Michigan is suffering today after fatal shooting. #POTUS, stop the bloodshed. #BackfireTrump https://t.co/Dn1lS5oz3Z

About

A Final Year Project for BEng in Software Engineering for Ulster University.

Contact

Thomas Franklin

franklin-t@ulster.ac.uk

Figure 13 /hashtags page illustrating the use of pie charts to provide details on rumours/non-rumours

The walk through outlined in this section, illustrates how a new Tweet from Twitter can make its way through the service in order to reach the audience on the frontend. The use of filters and ignore lists allows for control of the service to ensure only appropriate Tweets come through the service, as Twitter is a public service, there is a lot of advertisements etc. that are in large volumes as they make use of popular keywords/hashtags in order to reach a larger audience, by adding these known users/hashtags to ignore lists can prevent them from being processed by the system.

4.5 Consideration for Security

The project did not require any form of user login, which meant that authentication or authorisation was not a huge concern, as the project presents public data in an interactive way it would have been counter intuitive to hide it behind a login; however the most common attacks are designed to steal information, introduce vulnerabilities, and even cause damage to the behaviour of the software (Techopedia.com, 2018) and often come in the form of buffer/stack overflow, command injection and SQL Injections.

As discussed in section 3.3.1, the security against SQL Injection will be tackled with the use of prepared statements, as this is often the first form of defence against an SQL Injection attack, as it ensures that the attacker is not able to change the intent of the query (SQL Injection Prevention Cheat Sheet – OWASP, 2018).

The prevention of buffer/stack overflow attacks come down to the system design, as measures have been put in place so that rate limits have been set on user requests etc. in order to ensure that someone does not try and take down the service in the form of a denial of service attack, and by separating out the services, it means that if one service is targeted, then it can be isolated and prevent the spread to other services within the project.

To ensure the security of the service, monitoring will be in place and additional techniques can be implemented if deemed necessary, such as the use of honey pots, which is described as a computer system that is set up to act as a decoy to lure cyber attackers, and to detect, deflect or study attempts to gain unauthorized access to information systems (SearchSecurity, 2018).

5. System Verification

This section will provide the details and evidence of the testing of the project through using various testing strategies that are used within industry, as well as providing the results and the conclusion as to if the project meets the product requirements.

5.1 Verification Strategy

The project aimed to make the use of automated testing to cover the verification side of the project, how this was achieved was through what is commonly referred to as the “testing pyramid” illustrated in figure 14. The testing pyramid consists of three distinctive layers, where the bottom layer is the unit-tests, automated integration tests in the centre and finally at the top is the end-to-end testing, commonly referred to as UI testing (The Dojo, 2018).

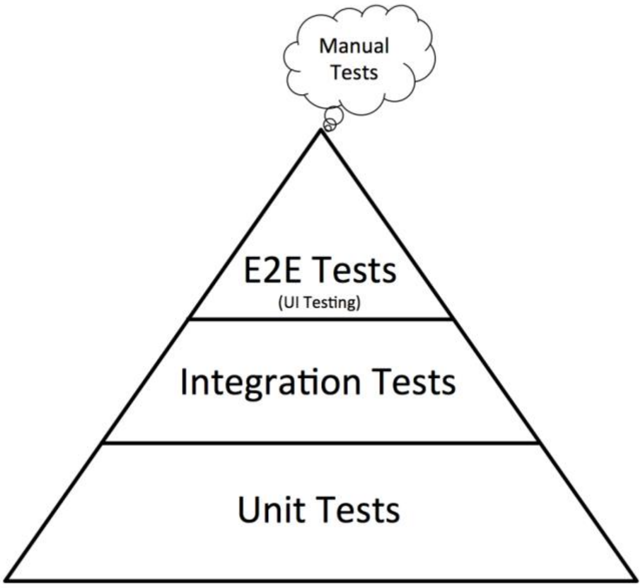


Figure 14 Automated Testing Pyramid (The Dojo, 2018)

Each layer is a representation of the volume of tests that should be covered, as unit tests are usually quick to run and less costly, they should form the majority of the test cases, while each layer gets progressively costlier and more time consuming to run hence why there should be fewer integration tests than unit tests, and even less UI tests.

(Software Testing Fundamentals, 2018) described unit tests as those that test individual units/components, while integration testing is where the units are combined and tested as a group; typically, every individual unit of the project could be tested at a unit level, while integration tests will be reserved for those critical components, such as handling database requests, etc.

For the project the aim is to test the core parts at a unit level through the use of JUnit[22] while integration tests will be for the classification process to ensure accuracy when saving to the database; although the project does have a frontend, there will be no automated UI testing, and this will be part of the manual test cases to ensure completeness.

5.2 System Verification Results

The results of the various automated verification tests which are executed on the project are provided in table 6, where there is a mixture of unit tests (UT.*), and integration tests (IT.*) and how they relate to the requirements set which are provided in section 2.2, while table 7 shows the manual tests which were performed on the system.

[22] JUnit is a simple framework to write repeatable unit tests in the Java programming language. It is an instance of the xUnit architecture for unit testing frameworks. (JUnit, 2018)

Table 6 Automated test scenarios

Test ID	Description	Test Name	Requirement Test	Pass/Fail
UT.01	Only a rumour or non-rumour label can be assigned	testThatRumourAndNonRumour_areTheOnlyLabels	FR.03	PASS
UT.02	Mapping of classification value to classification code	whenClassificationValueIsX_codeX_isReturned	FR.03	PASS
IT.01	Saving of a classified Tweet to database	whenItemIsProcessedAndHandled_itIsCorrectlySavedToDatabase	NF.03	PASS
UT.03	Classification label is used when verification label matches	whenClassificationIsSameAsVerification_classificationLabelIsUsed	NF.02	PASS
UT.04	Classification label is used when its weight is above the threshold	whenClassificationAboveThreshold_verificationValidationUsed	NF.02	PASS
UT.05	Verification label is used when classification weight below threshold	whenClassificationBelowThreshold_verificationValidationUsed	NF.02	PASS
UT.06	Test that the API correctly returns JSON objects	whenAPICallIsMade_responseWillBeJSON	FR.12	PASS
UT.07	Test the accuracy of classification	tenFoldCrossValidationTest (Weka)	NF.02	PASS
UT.08	Test the accuracy of classification	tenFoldCrossValidationTest (Mallet)	NF.02	PASS

Table 7 Manual test scenarios

Test ID	Description	Scenario	Requirement Tested	Expected Result	Actual Result	Pass/Fail
MT.01	User wants to check if a service is running	The pre-processor, queue reader and stream are not running	FR.02, NF.07, FR.11	Homepage should show that the pre-processor, queue reader and stream are not running	Homepage does show that these services are not running	PASS
MT.02	Pre-processor can be configured to use pre-processing	The configuration file for pre-processor for USE_PRE_PROCESSING key is set from false to true	NF.01	When new Tweets are processed the processed tweet should have the rules applied	New processed Tweets are missing URLs, #, @s etc.	PASS
MT.03	Tweet Stream will reconnect upon error	Stream service is running, and internet connection is lost	NF.05	Stream should reconnect upon regaining internet connection	Stream successfully reconnects when internet connection regained	PASS
MT.04	Configuration of Filter List to stream in to the service	The configuration file for stream for TWITTER_FILTER_LIST is altered	FR.06	Stream should be redeployed with the new filter	Stream is redeployed with the new filter	PASS
MT.05	User navigates to homepage of service	Can the user access the homepage	FR.10, FR.11	Frontend should display the system status information to the user	Frontend does display the system status information	PASS
MT.06	User navigates to users' page of the service	Can the user access the "users" page	FR.10	Frontend should display the top users results	Frontend does display the top users results	PASS
MT.07	User navigates to the hashtags page of the service	Can the user access the "hashtags" page	FR.10	Frontend should display the top hashtags results	Frontend does display the top hashtag results	PASS

5.3 Conclusion of Verification

The test scenarios provided in table 6 provide the core automated tests which were performed on the system, although if the project was to be restarted or given more time a larger emphasis would have been put in to the automated testing as most projects testing is left till the end which is unfortunate as it is vital to building the product right.

Whereas with table 7, which provides a list of some of the manual tests which were conducted on the system, given more time the project would have used an automated UI testing framework, such as Selenium[23] to conduct these tests. Of course, the list of manual tests could go on forever, which is why they have been condensed in to snippets outlining the core scenario which was conducted. In conclusion to the verification performed on the project, it could have done with more automated tests, however with the tests that have been conducted against the requirements the project is built right as it meets the requirements.

6. System Validation

This section will outline the strategy used for validating the project and ensuring that it meets the customer expectations, as well as document the results of the validation tests which have been conducted with the end-users identified in the strategy.

6.1 Validation Strategy

Validation is to be done at the end of the project and helps in determining if the system complies with the requirements, and tries to answer the question; "am I building the right product?" (What is Validation in Software Testing?, 2018).

In order to determine if the right product has been built, it was determined that the use of usability testing would be performed, in order to conduct the usability testing the following guidelines have been enforced (Loring B, 2018);

(i) Must be performed with representative end users of the product
(ii) Conducted in an actual or simulated use environment
(iii) Conducted with production equivalent units
(iv) Tasks must include primary functions, both frequent and infrequent tasks

The use of usability testing to validate the project helps to identify potential issues and problems, provides feedback direct from the target audience, and minimises the risk of failure (Experience UX, 2018). Table 8 shows the profiles of three potential end users of the system, although it would have been better to use more, for the size of the project it was determined that three would be enough providing they have different skill levels.

Table 8 Representative end users of the system for Usability Testing

Attributes	Participant 1	Participant 2	Participant 3
Age Group	16-21	35-40	45-70
Technical Ability	Moderate	High	Low
Hours spent using social media sites a week	6-10 hours	15+ hours	1-2 hours

In order to be consistent in what was being tested it was determined that the same laptop would be used to perform the tests for each user, and as usability testing requires evaluation the developer had to be there for the tests; which meant that it made sense to use the development laptop. Also, to ensure the system was validated correctly, each participant performed the same scenarios, the scenarios which had to be evaluated are listed in table 9 and 10, any scenario which is related to a particular requirement is also identified in the table.

[23] https://www.seleniumhq.org/

Table 9 Scenarios and the tasks for Usability testing

Scenario ID	Description	Tasks	Requirement
1	Main page observations	Access the homepage	NF.12
2	View system status	Ask participant if they can get an indication of the status of the system	NF.07, FR.02, FR.11
3	View data overview	Ask participant if they understand the data overview	NF.12
4	View details on filter list	Ask participant if there are any filter terms they would remove/add if they could	FR.14
5	Can users page be accessed	Ask the participant if they can see how to navigate to ‘users’	FR.10
6	Users page observations	Access the user page	NF.12
7	Can the participant select a user	Ask the participant to select any of the users	NF.12, FR.04
8	Information on users’ observations	Ask the participant what they understand about the information	NF.12
9	Can the participants see charts on the information	Ask the participant to select a chart	-
10	Is there any other information the participant would expect	Ask the participant for feedback on the information presented	-
11	Can the participant see the ‘raw data’	Ask the participant if they can easily view the raw data on the users’ results	NF.12
12	Are there any improvements that could be made to the raw data results	Ask the participant for feedback on the raw data	-
13	Are there any improvements which could be made to the users’ results page	Ask the participant for feedback on the page	NF.12
14	Can hashtags page be accessed	Ask the participant if they can see how to navigate to ‘hashtags’	FR.10
15	Hashtags page observations	Access the hashtags page	NF.12
16	Can the participant select a hashtag	Ask the participant to select any of the hashtags	NF.12, FR.05
17	Information on hashtags’ observations	Ask the participant what they understand about the information	NF.12
18	Can the participants see charts on the information	Ask the participant to select a chart	-
19	Is there any other information the participant would expect	Ask the participant for feedback on the information presented	-
20	Can the participant see the ‘raw data’	Ask the participant if they can easily view the raw data on the hashtags’ results	NF.12
21	Are there any improvements that could be made to the raw data results	Ask the participant for feedback on the raw data	-
22	Are there any improvements which could be made to the hashtags’ results page	Ask the participant for feedback on the page	NF.12

Table 10 Scenarios and the tasks for Usability Testing continued

Scenario ID	Description	Tasks	Requirement
23	Does the participant know how to perform a search	Ask the participant to see how they could perform a search	NF.12
24	Can a search be performed	Perform a search for the hashtag 'news'	FR.10
25	Search page observations	Ask the participant how the find the search results	NF.12
26	Can the participants see charts on the information	Ask the participant to select a chart	NF.12
27	Is there any other information the participant would expect	Ask the participant for feedback on the information presented	-
28	Are there any improvements which could be made to the search	Ask the participant for feedback on the page	NF.12
General Questions			
29	What is liked most	Ask the participant for feedback	-
30	What would the participant change	Ask the participant for feedback	-
31	What is liked least	Ask the participant for feedback	-
32	Are there any additional comments	Ask the participant for feedback	-

The scenarios were designed to cover the primary functions of the project, and to access the usability of the system as per the usability guidelines outlined in section 3.2.1; with the feedback from the usability testing improvements could be made to the system, the results of which are provided in section 6.2 in table 11, 12, 13 and 14.

6.2 System Validation Results

Table 11 Usability testing results for the three participants

	Observations		
Scenario	Participant 1	Participant 2	Participant 3
1	Liked how the site split the information in to columns	Was a bit surprised on the size of the stats	Wasn't sure why there was so much information on the page
2	Participant was quick to identify the services on the right and the colours indicating their status	Participant was quick to identify the services on the right and the colours indicating their status	Participant did not relate the colours on the right to the status of the systems
3	Participant didn't understand at first that the numbers represented the amount of data stored for the categories	Participant pointed out the stats immediately and understood what they meant	Participant required an explanation as to what the data meant
4	Participant took a while, but eventually realised that the filters currently on the system were more related to weather and climate	Participant suggested that the filter list is revised, and improvements could be made	Participant wondered if it was possible to change the filters live, explanation provided that the services would need restarted

Table 12 Usability testing results for the three participants continued

	Observations		
Scenario	Participant 1	Participant 2	Participant 3
5	Participant could identify the “users” link the navbar within 10 seconds	Participant had identified the “users” link in the navbar within 10 seconds	Participant had to be directed to the “users” link in the navbar after spending 20+ seconds searching
6	Participant was pleased with the speed in which the page loaded	Participant was pleased with the speed in which the page loaded	Participant appreciated how the navbar was the same
7	Participant related the radio buttons with selection on the right within 10 seconds	Participant immediately related the radio buttons as to how to select a user	Participant required assistance in selecting a user even after reading the help text
8	Participant understood the use of word clouds and how it relates to the information on the user selected	Participant suggested that the users name should be removed from the word cloud	Participant has never seen word clouds before and required an explanation
9	Participant could navigate between the charts within 10 seconds	Participant could navigate between the charts within 10 seconds	Participant could navigate between the charts within 20 seconds, they were distracted by the word cloud
10	Participant appreciated the use of bar and pie charts, and did not feel any other charts/visualisation aids would be needed	Participant did appreciate the use of pie and bar charts, however would have liked to see maybe a timeline or similar feature	Participant liked how the charts were interactive, and couldn’t think of any other visualisation aids that could be used
11	Participant was quick to spot the ‘raw data’ and liked how the results were limited per page	Participant was quick to spot the ‘raw data’ and liked how the results were limited per page	Participant was concerned that the table did not load fully, and had to explain that the results are 10 per page
12	Participant suggested highlighting rows which were rumours and those which were non-rumours to relate to the chart colours	Participant suggested including more details, such as when the Tweet was made, and possibly allowing you to filter by most recent	Participant suggested having a tip such as “Page 1 of x” to make it clearer
13	Participant was pleased with the page overall and its appearance	Participant would have expected more visualisation techniques to be used	Participant liked how the design was consistent with the homepage, however would expect more help in forms of tooltips etc.
14	Participant spotted the “hashtags” link in navbar quicker this time	Participant spotted the “hashtags” link in navbar quicker this time	Participant recognised the use of the navbar for navigating and was quick to spot it

Table 13 Usability testing results for the three participants continued

	Observations		
Scenario	Participant 1	Participant 2	Participant 3
15	Participant was again impressed with the speed and appreciated how the style of the page did not change	Participant was again impressed with the speed and appreciated how the style of the page did not change	Participant was pleased with the style remaining the same
16	Participant was quick to relate the radio buttons with selection of a term	Participant was quick to relate the radio buttons with selection of a term	Participant recognised the use of radio buttons to select a term
17	Participant understood the use of word clouds and how it relates to the information on the term selected	Participant understood the use of word clouds and how it relates to the information on the term selected	Participant was still unsure what word clouds were, but understand the use
18	Participant could navigate between the charts within 10 seconds	Participant could navigate between the charts within 10 seconds	Participant could navigate between the charts within 10 seconds
19	Participant suggested maybe additional charts could be added after closer inspection, however did not have any suggestions	Participant still suggested that more visualisation aids could be used, and again suggested the use of a timeline	Participant again appreciated how they were interactive
20	Participant was quick to spot the ‘raw data’ and liked how the results were limited per page	Participant was quick to spot the ‘raw data’ and liked how the results were limited per page	Participant understood the limits per page this time and appreciated it
21	Participant insisted again on the use of highlighting for the different classifications	Participant still wants to see additional information and allow filtering	Participant appreciates the use of pagination but would expect more help in how many results etc.
22	Participant really liked how the style of the page was maintained	Participant maintains the suggestions made for the ‘users’ page	Participant liked how the design was consistent with the homepage, however would expect more help in forms of tooltips etc.
23	Participant spotted the search bar in the navbar within 10 seconds	Participant spotted the search bar in the navbar within 10 seconds	Participant took around 15 seconds to relate the search bar to searching for results
24	Participant realised immediately the search bar could search both users and hashtags results	Participant realised immediately the search bar could search both users and hashtags results	Participant was unsure if the search was for hashtags
25	Participant appreciated how the search results layout was similar to the previous pages and helped with recognition	Participant appreciated how the search results layout was similar to the previous pages and helped with recognition	Participant appreciated how the search results layout was similar to the previous pages and helped with recognition

Table 14 Usability testing results for the three participants continued

	Observations		
Scenario	Participant 1	Participant 2	Participant 3
26	Participant immediately navigated between the charts	Participant immediately navigated between the charts	Participant immediately navigated between the charts
27	Participant liked how the pages were kept consistent, however for performing a search they would expect more information as you are searching a specific thing	Participant liked how the pages were kept consistent, however for performing a search they would expect more information as you are searching a specific thing	Participant appreciated the consistency in style
28	Participant suggested including more detail in the search results	Participant thought the use of one search for both users and hashtags was a smart use, however possibly include the search for particular terms	Participant thought the search could be made clearer as they were unsure if it was searching for what they wanted
General Questions			
29	Participant really liked how the style of the website was consistent as it made things easier to understand	Participant really liked the visualisation types used as it painted a picture on the data	Participant liked the use of pagination for raw data as it was less overwhelming
30	Participant would change the search as it is limited to users and hashtags, would like to search particular words	Participant would allow the ability to control the filters of the tables that present raw data	Participant would include more helpful tips for users like themselves
31	Participant did not like the limitations of the search	Participant did not like the limited information on the raw data and the fact it could not be filtered	Participant did not like how they required assistance to use the site
32	Participant overall enjoyed the aesthetics of the service, and apart from the limited search they were pleased with it	Participant like the implementation of the service and appreciated how it presented the information in visual ways	Participant does not think it would be a site they would use frequently

6.3 Conclusion of Validation

From the usability testing the following key points have been identified which could be used as the basis for enhancements to the service;

(i) Search needs enhancements to make it less restrictive
(ii) Raw data information could be increased to include more information on particular Tweets
(iii) Inclusion of more detailed visualisation aids
(iv) The sites usability could be improved by more meaningful tips
(v) Word cloud feature is not as relevant as intended

With these results, and to answer the question posed in section 6.1; “am I building the right product?” (What is Validation in Software Testing?, 2018) the product is the right product providing it is targeted at the right audience; the usability testing has shown that users which are not big users of social media, or have little technical abilities, such as participant 3 would not appreciate what the product is trying to achieve, this is a reason why picking a large representation of potential end users is important as it can help in identifying who would get the most use from the product.

Another way to validate that the right project has been built is to relate the results back to the usability guidelines from section 3.2.1, the system has been built right in terms of providing visibility of the system, being consistent and meeting standards, and allowing for recognition rather than recall as well as being aesthetically pleasing and minimalistic in its design; which are important as it allows for retention to the service from users.

6.4 Considerations for future work

The core part of the system is the classification process that occurs in the backend away from the users, while the frontend is an extension to the service to provide users with a way to visualise the data which has been processed by the system; with this in mind there are some improvements which could be made to the service which would not be possible in the current timeframe, section 6.4.1 outline some of the enhancements which could be made to the current system given more time as well as some enhancements which will be made as part of the validation techniques performed, while 6.4.2 are ideal features which could be made to the project, but are not within the scope of work.

6.4.1 Potential enhancements

(i) Ability to update the filter list that is filtered from Twitter live, rather than having to make the change offline and restart the service
(ii) Ability to see Tweets from their respective location on a map
(iii) Ability to filter the raw data in the tables
(iv) Improvements to the error reporting used for monitoring purposes
(v) Allow the search to search more than just users and hashtag terms
(vi) Improvements to the automated testing and improve the overall test coverage of the project

These enhancements unfortunately would not be possible within the current timeframe; however, the following improvements will be made to the service as a result of the usability testing as there is enough time to do them after discussion the stakeholders,

(i) Additional visualisation type, a timeline feature to show for a particular user/hashtag how many rumours/non-rumours were in the last 5 hours in 1-hour increments,
(ii) Additional columns added to the raw data
(iii) Help tips enhancements, such as how many results are returned in the raw data, and how the search works
(iv) Improvements to the help tips
(v) Redesign of word cloud as it slows down performance

With the stakeholders' approval it was also deemed that the requirement to allow users to report on appropriateness of a certain filter term/hashtag/username in the results (FR.14 table 2) was not feasible to complete in the remaining time and as a result it could be a potential enhancement to the service if work continued.

6.4.2 Potential Features

The enhancements identified in 6.4.1 are small tasks in comparison and would only require additional time to implement which is not currently available for the project deadline; however, the features identified below would require a considerable amount of code change and will change the system design drastically in order to implement the features;

(i) **User control:** give users the ability to login and set up their own filters and ignore lists which will be tracked on their own profile; this would require user authentication and authorisation and the backend system would need to allow for additional Twitter streams, queues and queue-readers etc. to be instantiated for each user which registers

(ii) **Expose the backend for public use:** currently the system uses its own API to retrieve data, however this could be exposed to public consumers that also make use of the data for their own purposes, i.e. companies could retrieve their own relevant data

These features would be things which improve the commercial aspect of the project, as currently the project is more in the lines of a proof of concept[24] and would not be ready for commercial use without the inclusion of the features identified above.

7. Conclusion and Reflection

This section will provide a conclusion to the development of the project along with the reflection from the developer in terms of the project was approached in terms of the appropriateness of the project plan along with the development considerations that have been made.

7.1 Critical Appraisal of Project

The aim of the project was to produce a service which could help in identifying new Tweets from Twitter as a rumour or non-rumour using machine learning. From the initial research in to the project there were no prior services which made use of natural language processing to detect rumours in text which is why the project should be considered the first of its kind.

In terms of the project itself, the developer wanted to ensure that the project was developed with maintainability and scalability in mind as it had to deal with thousands of new Tweets every minute. It has been a challenging task to ensure that the system could handle the amount of data without hindering the end-users experience, which is why the developer ensured the system design followed that of a micro service architecture to ensure that it was maintainable and easily scaled where required.

However, the project is not without fault, as the intentions for the project were to follow best practices which can be achieved through the following practices; test-first programming, rigorous/regular refactoring, continuous integration and keeping design simple (Version One, 2018).

In terms of the approach to the project, like a lot of projects it fell in to a testing trap where testing was left too late or not completed effectively, had a test-first/test-driven development approach been taken then these issues could have been avoided. Overall, the project has been completed following most of these standards, had there been more time or if the developer knew what they

[24] A demonstration, the purpose of which is to verify that certain concepts or theories have the potential for real-world application. (Techopedia.com, 2018)

do now when starting then more emphasis would have been put in to the testing of the project and ensuring that there was more test coverage.

7.2 Reflection of Project Plan

The original project plan had a planned end date of 19th April 2018, exactly one week before the project deadline. This was providing that there were no alterations required to the project, however after the project planning presentation held in December 2017 a choice was made to change the methodology as outlined in section 7.2.2, which resulted in development of the project starting later than intended and as a direct result the project was not completed by the original planned date.

In order to still complete the project before the deadline, the developer determined that not all of the original intentions would be possible. After careful consideration and assistance from the stakeholders, it was deemed that the original plans for the project had to be scaled down; this resulted in the removal of some intended features of the project, as outlined below;

(i) Post processing of the classification which could have allowed relationships between a location of a Tweet and its classification, i.e. visual map of the amount of rumours/non-rumours from particular parts of the world

(ii) Use of continuous integration[25] (CI) which would have made use of Travis CI or Jenkins, which are both widely used tools in the software industry for executing CI practices

(iii) Deployment of the service, in order that it was not ran from the development laptop the project plan was to make use of a cloud service such as Digital Ocean[26] to host the project

(iv) Continuous delivery to the selected online service, i.e. when changes are made, and the CI job is a success, the changes will automatically be deployed to the online service

(v) User feedback support, such as allowing users to provide feedback on appropriateness of a user/hashtag term in the service

Features (i) to (v) above were deemed to be desirable features by the stakeholders which is why they were removed from the plan in order to meet the deadline and if the project had room they could be added back in.

7.2.1 Appropriateness of initial time/effort estimation

From the project plan, it outlined estimates of the timeframe for the core parts of the project which identified that the processing part of the service would take approximately 50 days as this was deemed at the time to be the most challenging, 13 days for the API development and around 20 days for the frontend.

Although the processing part was still a challenging task as it was a completely new area to the developer, the original estimation was longer than what was required, while the developer felt the API and frontend would be the easier of the tasks they did in fact turn out to be more problematic than planned which has resulted in them taking longer than planned.

Aside from the overestimation on the effort for the processing part, the estimates were accurate considering, as the underestimation of the API development evened out the time/effort required for the project. With the knowledge gained now, it would be fair to suggest that more time and effort could have been put in to the API and Frontend

[25] developer's changes are validated by creating a build and running automated tests against the build (Atlassian, 2018)
[26] https://www.digitalocean.com/

development as these are two areas which required considerable more time than initially planned.

7.2.2 *Appropriateness of Software Methodology Used*

In the early stages of the project, it had the intentions to use the spiral model, as mentioned in section 7.2 the lifecycle was changed after the feedback from the presentation indicated spiral was not the best choice for the project.

The spiral model is an incremental lifecycle with an emphasis on risk, which the developer felt would be best suited for the project, upon feedback it became clear that the project although a new challenge did not process as much risk as was expected which is why a choice was made to change to extreme programming before the project kicked off.

Extreme programming is a software lifecycle methodology based on the development and delivery of small increments to functionality (Munassar N & Govardhan A, 2010). Figure 15 illustrates the basis of a cycle in the extreme programming lifecycle, where it begins with the selection of user stories for the release, which are then broken down and then the release is planned; for this project these tasks were performed in the sprint planning which was held at the end of every iteration before the next one begun.

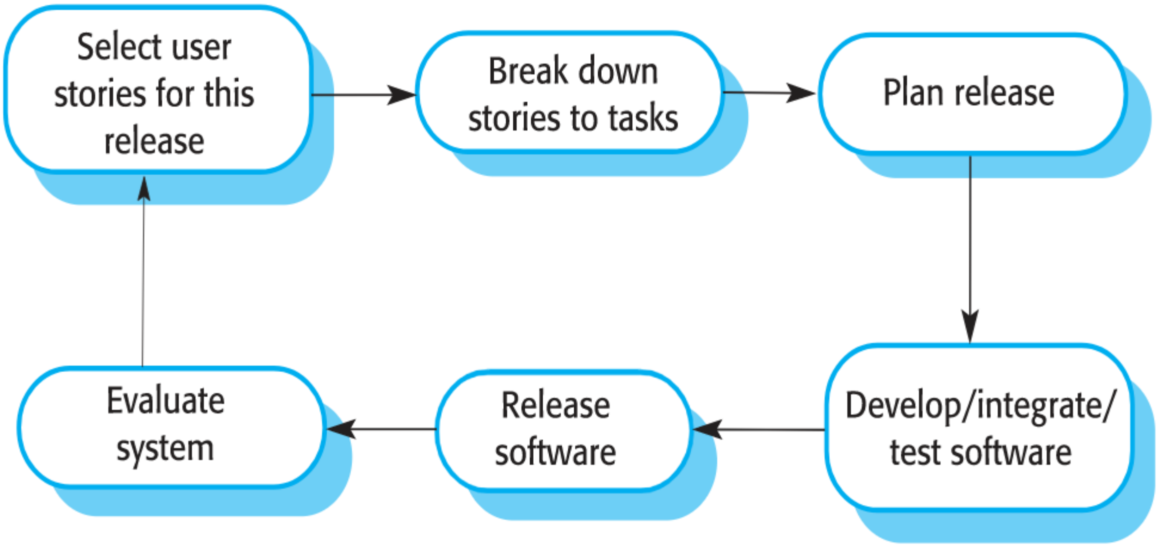
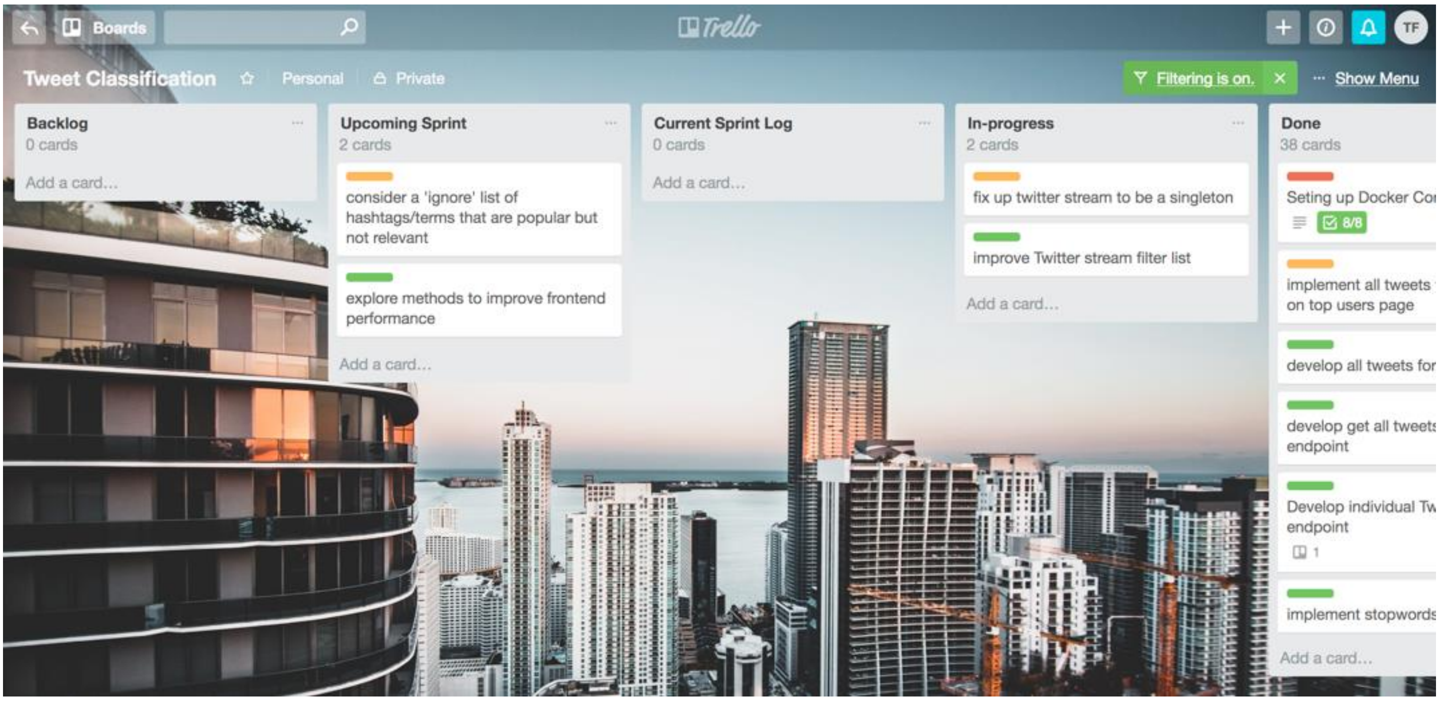

Select user stories for this release
Break down stories to tasks
Plan release
Develop/integrate/ test software
Release software
Evaluate system

Figure 15 Extreme programming release cycle (Munassar N & Govardhan A, 2010)

From the works of Munassar N & Govardhan A, 2010 had identified 10 practices which form the extreme programming lifecycle methodology, for this project the developer aimed to follow the practices as closely as possible, although not all were possible due to the size of the team; this meant pair programming was not at all possible as there was only one developer and the work could not be reviewed.

Selecting the wrong software lifecycle methodology could lead to disastrous results (Project Smart, 2018) which is why the lifecycle was changed to ensure that the project was not doomed before it begun.

With the change to extreme programming, the stakeholders felt it was an appropriate change as it gave them fortnightly feedback as basic functionality was delivered at the end of each release. As mentioned in section 4.1.1, the use of Trello was used to track the user stories of the project, where tasks could be broken down and assigned to sprints; the use of colour codes were used as illustrated in figure 16 to highlight the expected complexity of the task, where red indicates a complex task which will probably take the full 13 days of a release, orange is a medium task and would take half of a release to complete while green indicates an easy task and could typically be done in a few days.



Backlog
0 cards
Add a card…

Upcoming Sprint
2 cards
consider a 'ignore' list of hashtags/terms that are popular but not relevant
explore methods to improve frontend performance
Add a card…

Current Sprint Log
0 cards
Add a card…

In-progress
2 cards
fix up twitter stream to be a singleton
improve Twitter stream filter list
Add a card…

Done
38 cards
Seting up Docker Co
8/8
implement all tweets on top users page
develop all tweets for
develop get all tweets endpoint
Develop individual Tw endpoint
1
implement stopwords
Add a card…

Figure 16 Trello board indicating the stories per sprint

The colour coding helped in tracking and to determine velocity[27] which allowed the developer to monitor the project and estimate how long the project will take to complete based on the estimates of the remaining stories (Agile Alliance, 2018); it was with these estimates which allowed proactive steps to be taken to prioritise stories and ensure the most critical were completed earlier.

Using the extreme programming methodology was very suited for this project as it focuses on incremental delivery with an emphasis of small releases which suits the format, as each iteration delivered some value that could be reviewed and extended in sequent iterations; ensuring that the project meets the requirements and additional time allows for improvements to the service.

7.3 Conclusion

The project aimed to deliver a service which could classify new Tweets from Twitter as a rumour or non-rumour based on a Naïve Bayes classification problem, while being the first of its kind to see if natural language processing is capable of detecting rumours in text.

In terms of the accuracy of the classification, during the evaluation of the two trained classifiers; one using Weka library and the other using Mallet achieving accuracy score of 87% and 88% respectively – it seemed evident that natural language processing was a suited method in detecting rumours in text.

[27] At the end of each iteration, the team adds up effort estimates associated with user stories that were completed during that iteration (Agile Alliance, 2018)

However, in a real world scenario the results appear to be somewhat misleading, as using only the wording/phrases found in a Tweet to classify as a rumour does not appear to be the best application for the selected method, which is why the project was deemed to be a proof of concept to see how applicable the method is for rumour detection as it has proven success in the field of sentiment analysis.

Had the developer known earlier that natural language processing is not the most applicable method for rumour detection then other methods could have been explored and implemented, although in terms of application and delivery of the project, it delivered well executed code which can easily be extended and scaled to improve its applicability if more time could be allocated to the project.

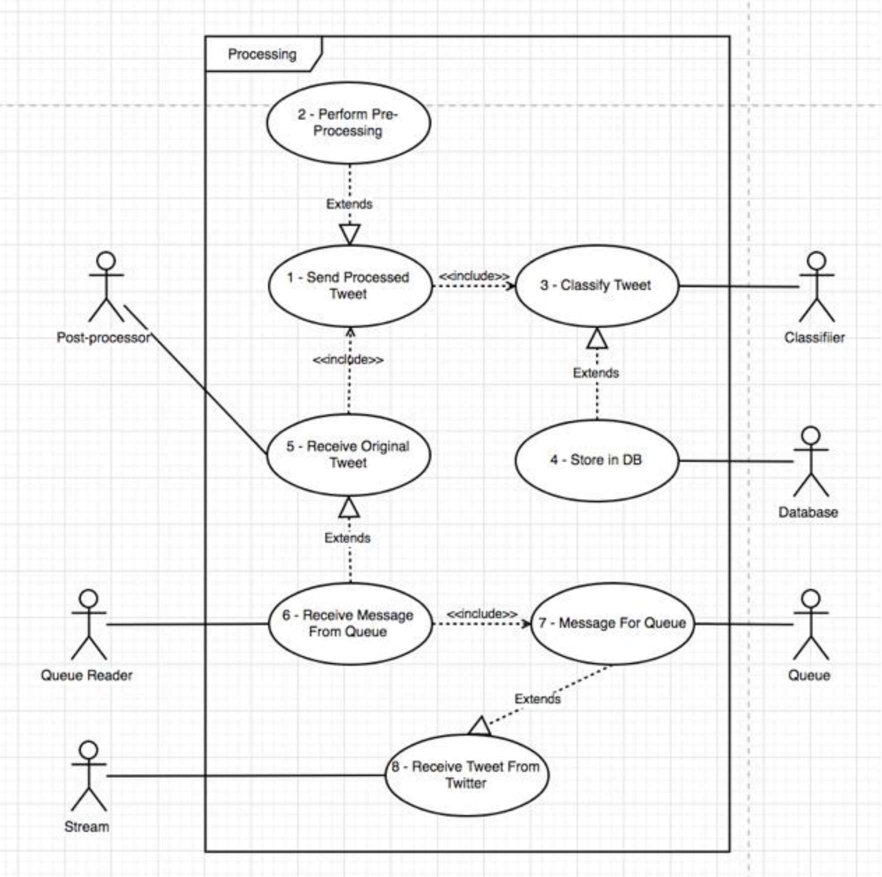
Overall the project was successful in the delivery of a service which used natural language to classify new Tweets as a rumour or non-rumour, making use of best practices and tools to aid in the development and implementation of a service which met the following core objectives;

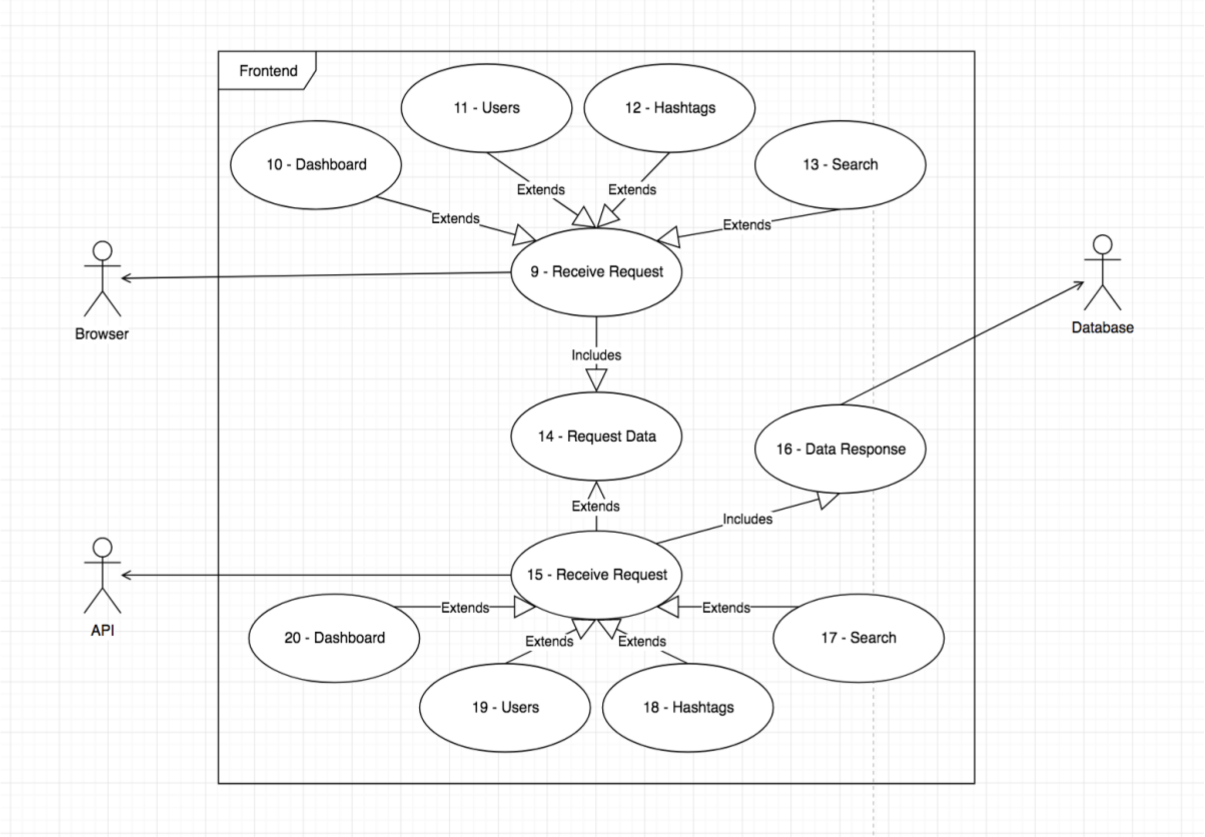
(i) The design and implementation of a system which continuously processes Tweets
(ii) The implementation of a classifier capable of assigning rumour or non-rumour label
(iii) A web service which uses visual methods of presenting the data to users
(iv) A structured RESTful API for allowing the processed data to be retrieved
(v) A robust service which is fault tolerant and can restart upon failure/error
(vi) Well-structured and self-documented code which is maintainable and can be reused for similar applications

Appendices

Appendix A: Final Requirements Formal Format

This appendix includes the use case diagrams were used to create the final list of requirements as seen in section 2.2, and the use of Volere shell was utilised to document them in a formal format and are provided below.





Requirement #:	FR.01	**Requirement Type:**	Functional	**Event/Use Case #:**	1
Description:	The system shall assign a rumour or non-rumor label to a processed Tweet				
Rationale:	To be able to assign a classification label to a particular Tweet after processing has been applied				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	If processing is enabled then the following rules should be applied: (1) lower case Tweet, (2) replace # at start of hashtags, (3) replace @ at start of usernames, (4) remove URLs, (5) remove double of more spaces, (6) remove new lines/character returns, remove trailing whitespace				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		1
Priority:	High		**Conflicts:**		FR-2
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.01	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	2
Description:	The system shall be configurable to account for the needs of the system				
Rationale:	To be able to control the systems services, and enable certain filters, processing techniques				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Configuration files for each service which are deployed at run time				
Customer Satisfaction:	3		**Customer Dissatisfaction:**		3
Priority:	Medium		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.02	**Requirement Type:**	Functional	**Event/Use Case #:**	1
Description:	The system shall provide details on if it is running				
Rationale:	To be able to check if the services are operational				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Various endpoints and checks to ensure services are functional				
Customer Satisfaction:	2		**Customer Dissatisfaction:**		1
Priority:	Low		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.03	**Requirement Type:**	Functional	**Event/Use Case #:**	3
Description:	The classifier shall only be able to assign a rumour or non-rumour label				
Rationale:	As the system is classification of rumour or non-rumour, by ensuring these are the only labels it will help in the classification process				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Checks to ensure the classifier only knows about rumours and non-rumours				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.02	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	3
Description:	The system shall make use of a two-stage classification process				
Rationale:	The system should use a verification classification to ensure the accuracy is as high as possible				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Use of two classification frameworks				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.03	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	4
Description:	The system shall accurately store the data to the database				
Rationale:	The system has to ensure consistency and reliability				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Verification of data results				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.04	**Requirement Type:**	Functional	**Event/Use Case #:**	4
Description:	The system shall be able to link back a Tweet to a particular user				
Rationale:	The system has to ensure traceability between Tweets and users to allow for the data to be presented correctly				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Validation of returned data				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.05	**Requirement Type:**	Functional	**Event/Use Case #:**	4
Description:	The system shall be able to link back a Tweet to a particular hashtag				
Rationale:	The system has to ensure traceability between Tweets and hashtags to allow for the data to be presented correctly				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Validation of returned data				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.06	**Requirement Type:**	Functional	**Event/Use Case #:**	8
Description:	The system shall be configurable with the filter list				
Rationale:	The system has to ensure that it has control over what Tweets get streamed in to the service				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Validation of retrieved data				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.04	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	8
Description:	The system shall only add Tweets that are not a retweet to the Queue				
Rationale:	As Retweets are repeated Tweets, processing them more than once will be counterproductive				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Check to ensure no retweets are processed				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		2
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.05	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	8
Description:	The system shall be robust enough to restart on failure				
Rationale:	As the service should be constantly streaming Tweets, it is important that it reconnects if there is ever an error				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Using appropriate libraries to ensure reliability				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.06	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	6
Description:	The system shall ensure that the message is valid from the queue				
Rationale:	To ensure that integrity of the system is kept, and the correct message is processed				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Serialise the message in to a known object				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.07	**Requirement Type:**	Functional	**Event/Use Case #:**	6
Description:	The system shall be able to report on successes/failures				
Rationale:	To monitor the performance of the system				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Use logging to output events				
Customer Satisfaction:	2		**Customer Dissatisfaction:**		3
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.08	**Requirement Type:**	Functional	**Event/Use Case #:**	6
Description:	The system shall remove/discard any items that are not a Tweet object				
Rationale:	To ensure that only relevant information is processed and kept				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Serialisation of objects to ensure correctness				
Customer Satisfaction:	2		**Customer Dissatisfaction:**		3
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.09	**Requirement Type:**	Functional	**Event/Use Case #:**	5
Description:	The system shall only handle Tweet objects from the queue reader				
Rationale:	To ensure that only relevant information is processed and kept				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Serialisation of objects to ensure correctness				
Customer Satisfaction:	2		**Customer Dissatisfaction:**		3
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.09	**Requirement Type:**	Functional	**Event/Use Case #:**	5
Description:	The system shall only handle Tweet objects from the queue reader				
Rationale:	To ensure that only relevant information is processed and kept				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Serialisation of objects to ensure correctness				
Customer Satisfaction:	2		**Customer Dissatisfaction:**		3
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.10	**Requirement Type:**	Functional	**Event/Use Case #:**	9
Description:	The system shall be able to receive user requests				
Rationale:	To ensure that users can connect to the service				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Routes to match home, users, hashtags and search				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.11	**Requirement Type:**	Functional	**Event/Use Case #:**	10
Description:	The system shall be able to display status information when user navigates to homepage				
Rationale:	To ensure that users can see the status of the system				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Homepage matches the “dashboard” design				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.07	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	10
Description:	The system shall only display relevant information to the status of the system				
Rationale:	To ensure that users only information about the status that are relevant, and errors/exceptions should be kept private				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Homepage matches the “dashboard” design				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.08	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	16
Description:	The system shall be free from SQL injection attempts				
Rationale:	To ensure that the database and its data is secure				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Implementing store procedures and other data design considerations				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.09	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	17
Description:	The system shall be secure				
Rationale:	To ensure that a user does not maliciously try and retrieve data				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Ensuring that data entered to the search is not directly sent to the database				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.10	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	Sep-15
Description:	The system shall be able to respond to user requests within 15 seconds				
Rationale:	To ensure that a user does not get bored waiting on a request				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Optimising the code to ensure it is as efficient as possible				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.12	**Requirement Type:**	Functional	**Event/Use Case #:**	17,18,19,20
Description:	The system shall return responses in JSON format				
Rationale:	To ensure that the frontend can handle the response and display the information to the user				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Making use of serialisation libraries to return JSON objects				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.11	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	14
Description:	Data requests from the Frontend shall be executed directly to the API				
Rationale:	To ensure that the frontend can communicate directly to the API				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Making use of Docker connectivity				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.13	**Requirement Type:**	Functional	**Event/Use Case #:**	10,11,12,13
Description:	The system shall provide help to the users				
Rationale:	To ensure that the system is easy to use and informative				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Making use of tooltips through appropriate libraries				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	FR.14	**Requirement Type:**	Functional	**Event/Use Case #:**	11,12
Description:	The system shall allow users to report particular terms				
Rationale:	In case a Tweet that doesn't match the filters that needs to be added				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Internal form which can be submitted				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

Requirement #:	NF.12	**Requirement Type:**	Non-Functional	**Event/Use Case #:**	10,11,12,13
Description:	The system shall be pleasing to the eye				
Rationale:	As the site is how users will interact, it should be pleasant to look at				
Originator:	Thomas Franklin - Developer				
Fit Criterion:	Use of design best practises and HCI considerations				
Customer Satisfaction:	5		**Customer Dissatisfaction:**		5
Priority:	High		**Conflicts:**		None
Supporting Material:	None				
History:	Version 1 - January 2018				

1.0 Transfer and Application of Student Knowledge and Skills

1.1 Purpose

This section will highlight how the University has helped shape my future, through the modules which were offered to the opportunities which I have had due to attending *Ulster University*.

1.2 Notable Modules

In first year I got an introduction to programming through *Programming I* which taught me the fundamentals of Object Orientated Programming, and without these foundations I do not think that I would be as strong of a programmer as I currently am. Another important module in first year that has helped is the *Introduction to Databases* module – which taught me the benefits and uses of a relational database such as MySQL.

Leading on to second year, I got to experience more programming through *Programming II* and *Programming III*, *Programming II* lead on to more advanced topics within the programming language *Java* where I got to learn more about the principles of Object Orientated Programming, such as encapsulation, inheritance etc. while with *Programming III* it introduced me to a new programming language, *C#*, which is also an Object Orientated Language which meant that the skills learnt were transferable to other programming languages.

Second year was a continuation on the foundations that had been built in first year, and I know for a fact that if I did not have the opportunity to complete first year I would have been at a huge disadvantage come second year, as the strong foundation which I built in first year would not be there to help.

I also got to experience more advanced uses of SQL databases through *Database Engineering* module, which taught me about more advanced uses such as optimising queries for faster retrieval, the importance of having strong relations between tables etc.

1.3 Notable Achievements

In first year I was awarded an award for best module performance in *Programming I* for *BEng Software Engineering* which was awarded by *Liberty IT* where I received a cheque and plaque, as well as this I received a certificate for achieving above 70% average in first year.

To continue on with the achievements, in second year I again received a certificate for averaging above 70% and as well as this I got a prize for SQS Testing which was due to “highest differential between student’s average mark for the year and average mark for the cohort”.

During my placement I also got nominated for an award by my supervisor, unfortunately I was not successful, although I still received a certificate for being nominated.

It would not have been possible without the continued support received from the *Ulster University* to have received these awards to date.

1.4 Extra-Curricular Activities

From the opportunities I have had and skills I’ve developed in my course I have had the chance to attend a few *Hackathons* with other students from *Ulster University*, the first one I attended was for the charity *Marie Curie* known as *Mariethon* which I took part in during my second year, which the team and I won even though none of us had prior *hackathon* experience.

A second one I attended was known as *Hack the Hub* operated by *PWC* which I attended during my placement year. This was more technical than the first one, as the first one I attended was more about idea generation that producing something – the *Hack the Hub* was a lot more challenging.

It was these extra-curricular activities which has helped me in generating ideas and working under pressure, and it would not have been possible without the knowledge which I have learnt from attending *Ulster University*.

1.5 Placement Opportunity

During my second year I had to try and secure a year of industrial experience in Software Engineering to be completed during my third year, with the achievements and the good grades achieved during my first year at *Ulster University* helped me in securing a placement at *Kainos* which is deemed as one of the top IT employer in *Belfast*.

During the placement year it helped in strengthening the skills learnt during my first and second year at *Ulster University*. My time at *Ulster University* helped by building strong foundations, while my time at *Kainos* helped in advancing my skills. While on placement I had the opportunity to work on one of the largest projects which *Kainos* had at the time, which was really good, and I developed a lot of new skills by working with technologies which are not currently taught within my degree, such as Amazon Web Services, NoSQL databases, etc.

The project I was on was operated from *Nottingham*, which meant I had to travel weekly Monday to Thursday each week for the year, this was a good experience as it taught me a lot of life skills and some independence, working on a client site is a different way of working as you need to be mindful that you are working alongside people who may not be technically minded, so you need to be considerate when explaining something which you may find easy.

1.6 Conclusion

The combination of the modules, achievements and placement mentioned above has gave me the skills and knowledge required to complete my final year project which is worth 25% of my degree which makes it very important to me to do well in.

2.0 The Project Experience

2.1 Purpose

This section will highlight my project and the experience I have had so far and how my time at *Ulster University* and my placement year at *Kainos* has assisted in the development of my project.

2.2 My Project

For my project I decided to do *Topic Classification in Social Media* which uses a combination of Machine Learning and Natural Language Processing to *classify Tweets* from *Twitter* as either a rumour or non-rumour which will help detecting news in social media which may be considered fake before it has a chance to spread and wrongly influence users.

The reasoning for doing this project was something I always wanted to do was something which involved *Artificial Intelligence* and the in recent years social media has become increasingly popular in spreading incorrect information to the masses. The architecture of the system is developed in *Java* with the use of *JQuery*, *Javascript*, *HTML, and CSS* for the front end, as the final project will be a website which will highlight topics which are potentially considered rumours – whereas *Java* will be used for the backend processing of the *Tweets* and the classification process.

The modules in first and second year ultimately drove my decision to use *Java* as it is the language I had most experience with, while the use of frontend technologies mentioned above was driven by the use of these technologies during my placement year and also through *Interactive Web Computing* – which was a module completed in my first semester of final year at *Ulster University*.

2.3 Project Status

I am currently finishing of the backend processing of my project, I have completed the pre-processing of the *Tweets* and the classification, I now just need to save the results to my MySQL database for retrieval from the frontend which will be developed in the coming weeks.

In order to keep track of my project I have been using management techniques developed during my final year modules such as *Formal Requirements Specification and Software Engineering Management* – as well as this the skills developed during placement of working in an Agile development team has allowed me to keep track of my work and report on it effectively which will helped when writing my final report.

Overall my project is currently on track and most of the development should be completed in the coming weeks which will allow me time to perform some validation testing with my stakeholders and finish off the final sections of my final report.

2.4 Conclusion

The combination of the skills developed on my placement year and the modules offered to me by *Ulster University* has driven the decisions made for my project – and with these decisions they have been influenced by my desire to do well on my project.

3.0 Preparation for Employer Interview

3.1 Purpose

This section will highlight the career options made available to be through *Ulster University* and my placement employer *Kainos*, while describing how a combination of the skills learnt in University have prepared me for working a Graduate Software Engineer.

3.2 Opportunities

Upon completing my placement year with *Kainos*, they offered me a Graduate contract on the condition that I graduated with at least a 2:1 – this opportunity came from the hard work which I put in during my placement year. I provisionally accepted this offered at the time and signed the contract.

During my final year I decided to apply for a few graduate jobs which appealed to my interests, as working at *Kainos* which primarily deals with consultancy work requires frequent travel if you want to work on interesting projects, which no longer appeals to me as I experienced it during my placement year and did not like the lifestyle of weekly travel and it had a negative impact on my health – which meant I was on the lookout for more product related companies.

From the 3 or 4 applications which I had submitted I attended an interview at a company called *Puppet* that has a *Belfast* office, this company is a product-based company which is what I want, and I have received a graduate contract for them as well on the condition I graduate with at least a 2:1.

This means I currently have two graduate opportunities available to me, neither of which would have been possible without the skills and experience I developed during my time at *Ulster University* and the placement year which I had which also would not have been possible had I not attended *Ulster University*.

3.3 Conclusion

Before Christmas of my final year I had already received two graduate contracts, which ultimately was a result of my own hard work and dedication alongside the skills and experience developed during my time at *Ulster University*.

4.0 Conclusion of Showcase Report

The modules, achievements and skills discussed in this report are what has lead me to this stage of my degree where my final year project is nearing an end, and I have graduate opportunities available to me when I graduate which is all due to the support received from the staff I have had the pleasure to learn from at *Ulster University*.

Bibliography:

About | What is Trello?. 2018. About | What is Trello?. [ONLINE] Available at: https://trello.com/about. [Accessed 11 April 2018].

Agile Alliance. 2018. What is Velocity in Agile? | Agile Alliance. [ONLINE] Available at: https://www.agilealliance.org/glossary/velocity/#q=~(filters~(postType~(~'page~'post~'aa_book~'aa_event_session~'aa_experience_report~'aa_glossary~'aa_research_paper~'aa_video)~tags~(~'velocity))~searchTerm~'~sort~false~sortDirection~'asc~page~1). [Accessed 23 April 2018].

Atlassian. 2018. Continuous integration vs. continuous delivery vs. continuous deployment |. [ONLINE] Available at: https://www.atlassian.com/continuous-delivery/ci-vs-ci-vs-cd. [Accessed 20 April 2018].

Analytics Vidhya. 2018. 6 Easy Steps to Learn Naive Bayes Algorithm (with code in Python) . [ONLINE] Available at: https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/. [Accessed 10 April 2018].

Apache Tomcat Project. 2018. Apache Tomcat® - Welcome!. [ONLINE] Available at: http://tomcat.apache.org/. [Accessed 07 April 2018].

Ben Shneiderman. 1997. Designing the User Interface: Strategies for Effective Human-Computer Interaction (3rd ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Ben W. Medlock, Investigating classification for natural language processing tasks, Tech. Report UCAM-CL-TR-721, University of Cambridge, Computer Laboratory, June 2008.

Beth Loring. 2018. Differentiating Between Validation Testing and Other Usability Testing. [ONLINE] Available at: https://www.farmpd.com/Farm-Blog/bid/55836/Differentiating-Between-Validation-Testing-and-Other-Usability-Testing. [Accessed 17 April 2018].

Bloomberg.com. 2018. Fake News, Trump and the Pressure on Facebook: QuickTake Q&A - Bloomberg. [ONLINE] Available at: https://www.bloomberg.com/news/articles/2016-11-25/fake-news-trump-and-the-pressure-on-facebook-quicktake-q-a. [Accessed 07 April 2018].

Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? sentiment classification using machine learning techniques. In Proceedings of the ACL-02 conference on Empirical methods in natural language processing - Volume 10 (EMNLP '02), Vol. 10. Association for Computational Linguistics, 79-86 [ONLINE] Available at: http://delivery.acm.org/10.1145/1120000/1118704/p79- pang.pdf [Accessed 11 October 2017].

Bootstrap 4 Get Started. 2018. Bootstrap 4 Get Started. [ONLINE] Available at: https://www.w3schools.com/bootstrap4/bootstrap_get_started.asp. [Accessed 11 April 2018].

Building RESTful Web Services with JAX-RS - The Java EE 6 Tutorial. 2018. Building RESTful Web Services with JAX-RS - The Java EE 6 Tutorial. [ONLINE] Available at: https://docs.oracle.com/javaee/6/tutorial/doc/giepu.html. [Accessed 11 April 2018].

Checkmarx. 2018. The Importance of Database Security and Integrity. [ONLINE] Available at: https://www.checkmarx.com/2016/06/24/20160624the-importance-of-database-security-and-integrity/. [Accessed 09 April 2018].

Creately Blog. 2018. Use Case Diagram Relationships Explained with Examples - Creately Blog. [ONLINE] Available at: https://creately.com/blog/diagrams/use-case-diagram-relationships/. [Accessed 10 April 2018].

Database Trends and Applications. 2018. Bad Database Standards Can Cause Performance Problems - Database Trends and Applications . [ONLINE] Available at: http://www.dbta.com/Columns/DBA-Corner/Bad-Database-Standards-Can-Cause-Performance-Problems-91192.aspx. [Accessed 09 April 2018].

DigitalOcean. 2018. How To Create a New User and Grant Permissions in MySQL | DigitalOcean. [ONLINE] Available at: https://www.digitalocean.com/community/tutorials/how-to-create-a-new-user-and-grant-permissions-in-mysql. [Accessed 09 April 2018].

Differences between Java EE and Java SE - Your First Cup: An Introduction to the Java EE Platform. 2018. Differences between Java EE and Java SE - Your First Cup: An Introduction to the Java EE Platform. [ONLINE] Available at: https://docs.oracle.com/javaee/6/firstcup/doc/gkhoy.html. [Accessed 11 April 2018].

Docker. 2018. Docker - Build, Ship, and Run Any App, Anywhere. [ONLINE] Available at: https://www.docker.com/. [Accessed 11 April 2018].

Docker. 2018. What is a Container | Docker. [ONLINE] Available at: https://www.docker.com/what-container. [Accessed 07 April 2018].

Experience UX. 2018. What is wireframing | Experience UX. [ONLINE] Available at: https://www.experienceux.co.uk/faqs/what-is-wireframing/. [Accessed 09 April 2018].

Experience UX. 2018. What is usability testing? | Experience UX. [ONLINE] Available at: https://www.experienceux.co.uk/faqs/what-is-usability-testing/. [Accessed 17 April 2018].

Extreme Programming: A Gentle Introduction.. 2018. Extreme Programming: A Gentle Introduction.. [ONLINE] Available at: http://www.extremeprogramming.org/. [Accessed 09 April 2018].

Facebook and Twitter unveil plans to tackle fake news | News | Ecommerce Week. 2018. Facebook and Twitter unveil plans to tackle fake news | News | Ecommerce Week. [ONLINE] Available at: http://www.ecommerceweek.co.uk/news/683/facebook-and-twitter-unveil-plans-to-tackle-fake-news/. [Accessed 07 April 2018].

Fake news definition and meaning | Collins English Dictionary. 2018. Fake news definition and meaning | Collins English Dictionary. [ONLINE] Available at: https://www.collinsdictionary.com/dictionary/english/fake-news. [Accessed 07 April 2018].

Git. 2018. Git. [ONLINE] Available at: https://git-scm.com/. [Accessed 11 April 2018].

Git - About Version Control. 2018. Git - About Version Control. [ONLINE] Available at: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control. [Accessed 11 April 2018].

GitHub. 2018. GitHub - google/guice: Guice (pronounced 'juice') is a lightweight dependency injection framework for Java 6 and above, brought to you by Google.. [ONLINE] Available at: https://github.com/google/guice. [Accessed 14 April 2018].

GitHub. 2018. GitHub - jknack/handlebars.java: Logic-less and semantic Mustache templates with Java. [ONLINE] Available at: https://github.com/jknack/handlebars.java. [Accessed 14 April 2018].

Go, Bhayani, Huang, 2009. Twitter Sentiment Classification using Distant Supervision. Final. Computer Science: Stanford University.

Gradle Build Tool. 2018. Gradle Build Tool. [ONLINE] Available at: https://gradle.org/. [Accessed 11 April 2018].

Java EE - Java Platform, Enterprise Edition | Oracle United Kingdom. 2018. Java EE - Java Platform, Enterprise Edition | Oracle United Kingdom. [ONLINE] Available at: https://www.oracle.com/uk/java/technologies/java-ee.html. [Accessed 11 April 2018].

Jersey. 2018. Jersey. [ONLINE] Available at: https://jersey.github.io/. [Accessed 11 April 2018].

JSON. 2018. JSON. [ONLINE] Available at: https://www.json.org/. [Accessed 11 April 2018].

jQuery Foundation - jquery.org. 2018. jQuery. [ONLINE] Available at: https://jquery.com/. [Accessed 14 April 2018].

JUnit - About. 2018. JUnit - About. [ONLINE] Available at: https://junit.org/junit4/. [Accessed 16 April 2018].

Karl Wiegers. 1999. First things first. Softw. Dev. 7, 9 (September 1999), 48-53.

Kate Conger. 2018. Here Are 14 Russian Ads That Ran on Facebook During The 2016 Election. [ONLINE] Available at: https://gizmodo.com/here-are-14-russian-ads-that-ran-on-facebook-during-the-1820052443. [Accessed 07 April 2018].

Kirill Fakhroutdinov. 2018. UML Information Flow Diagrams - Overview of Graphical Notation. [ONLINE] Available at: https://www.uml-diagrams.org/information-flow-diagrams.html. [Accessed 09 April 2018].

Kirill Fakhroutdinov. 2018. Use case diagrams are UML diagrams describing units of useful functionality (use cases) performed by a system in collaboration with external users (actors).. [ONLINE] Available at: https://www.uml-diagrams.org/use-case-diagrams.html. [Accessed 09 April 2018].

LeadingAgile. 2018. Works on my Machine -LeadingAgile. [ONLINE] Available at: https://www.leadingagile.com/2017/03/works-on-my-machine/. [Accessed 20 April 2018].

Lee K, Palsetia D, Narayanan R, Patwary M, Agrawal A, and Choudhary A, 2011. Twitter Trending Topic Classification. Final. Department of Electrical Engineering and Computer Science: Northwestern University.

Lifewire. 2018. Choosing a Database for Your Organization. [ONLINE] Available at: https://www.lifewire.com/choosing-a-database-for-your-organization-1019601. [Accessed 11 April 2018].

MALLET homepage. 2018. MALLET homepage. [ONLINE] Available at: http://mallet.cs.umass.edu/. [Accessed 10 April 2018].

Mark Otto, Jacob Thornton, and Bootstrap contributors. 2018. Bootstrap · The most popular HTML, CSS, and JS library in the world.. [ONLINE] Available at: https://getbootstrap.com/. [Accessed 11 April 2018].

Meyer, Robinson. 2018. Huge MIT Study of ‘Fake News’: Falsehoods Win on Twitter - The Atlantic. [ONLINE] Available at: https://www.theatlantic.com/technology/archive/2018/03/largest-study-ever-fake-news-mit-twitter/555104/. [Accessed 07 April 2018].

MuleSoft. 2018. What is an API? (Application Programming Interface) | MuleSoft. [ONLINE] Available at: https://www.mulesoft.com/resources/api/what-is-an-api. [Accessed 11 April 2018].

Munassar N, Govardhan A, 2010. A Comparison Between Five Models Of Software Engineering. IJCSI International Journal of Computer Scie nce Issues, [Online]. Vol. 7, Issue 5, pp, 95-101. Available at: http://www.ijcsi.org/papers/7-5-94-101.pdf [Accessed 23 April 2018].

MySQL. 2018. MySQL. [ONLINE] Available at: http://www.oracle.com/technetwork/database/mysql/index.html. [Accessed 11 April 2018].

MySQL :: MySQL 5.7 Reference Manual :: 6.3.5 Setting Account Resource Limits. 2018. MySQL :: MySQL 5.7 Reference Manual :: 6.3.5 Setting Account Resource Limits. [ONLINE] Available at: https://dev.mysql.com/doc/refman/5.7/en/user-resources.html. [Accessed 09 April 2018].

MySQL :: MySQL 5.7 Reference Manual :: 13.7.1.4 GRANT Syntax. 2018. MySQL :: MySQL 5.7 Reference Manual :: 13.7.1.4 GRANT Syntax. [ONLINE] Available at: https://dev.mysql.com/doc/refman/5.7/en/grant.html. [Accessed 09 April 2018].

MySQL :: MySQL 5.7 Reference Manual :: 13.5 Prepared SQL Statement Syntax. 2018. MySQL :: MySQL 5.7 Reference Manual :: 13.5 Prepared SQL Statement Syntax. [ONLINE] Available at: https://dev.mysql.com/doc/refman/5.7/en/sql-syntax-prepared-statements.html. [Accessed 09 April 2018].

Nick Babich. 2018. Golden Rules of User Interface Design. [ONLINE] Available at: http://babich.biz/golden-rules-of-user-interface-design/. [Accessed 09 April 2018].

Nielsen Norman Group. 2018. 10 Heuristics for User Interface Design: Article by Jakob Nielsen. [ONLINE] Available at: https://www.nngroup.com/articles/ten-usability-heuristics/. [Accessed 09 April 2018].

NGINX. 2018. Microservices at Netflix: Lessons for Architectural Design. [ONLINE] Available at: https://www.nginx.com/blog/microservices-at-netflix-architectural-best-practices/. [Accessed 07 April 2018].

Oxford Dictionaries | English. 2018. interface | Definition of interface in English by Oxford Dictionaries. [ONLINE] Available at: https://en.oxforddictionaries.com/definition/interface. [Accessed 11 April 2018].

PHEME. 2018. PHEME dataset of rumours and non-rumours. [ONLINE] Available at: https://figshare.com/articles/PHEME_dataset_of_rumours_and_non-rumours/4010619. [Accessed 10 April 2018].

POST statuses/filter — Twitter Developers. 2018. POST statuses/filter — Twitter Developers. [ONLINE] Available at: https://developer.twitter.com/en/docs/tweets/filter-realtime/api-reference/post-statuses-filter.html. [Accessed 07 April 2018].

Project Smart. 2018. Which Life Cycle Is Best for Your Project?. [ONLINE] Available at: https://www.projectsmart.co.uk/which-life-cycle-is-best-for-your-project.php. [Accessed 23 April 2018].

RabbitMQ - Messaging that just works. 2018. RabbitMQ - Messaging that just works. [ONLINE] Available at: https://www.rabbitmq.com/. [Accessed 11 April 2018].

Science | AAAS. 2018. Fake news spreads faster than true news on Twitter—thanks to people, not bots | Science | AAAS. [ONLINE] Available at: http://www.sciencemag.org/news/2018/03/fake-news-spreads-faster-true-news-twitter-thanks-people-not-bots. [Accessed 07 April 2018].

SearchDataManagement. 2018. Three indicators that could signal database performance issues. [ONLINE] Available at: https://searchdatamanagement.techtarget.com/feature/Three-indicators-that-could-signal-database-performance-issues. [Accessed 09 April 2018].

SearchMicroservices. 2018. What is RESTful API? - Definition from WhatIs.com. [ONLINE] Available at: https://searchmicroservices.techtarget.com/definition/RESTful-API. [Accessed 07 April 2018].

SearchSecurity. 2018. What is honeypot (honey pot)? - Definition from WhatIs.com. [ONLINE] Available at: https://searchsecurity.techtarget.com/definition/honey-pot. [Accessed 15 April 2018].

SearchSoftwareQuality. 2018. What is user story? - Definition from WhatIs.com. [ONLINE] Available at: https://searchsoftwarequality.techtarget.com/definition/user-story. [Accessed 11 April 2018].

Shneiderman's "Eight Golden Rules of Interface Design" | Design Principles FTW. 2018. Shneiderman's "Eight Golden Rules of Interface Design" | Design Principles FTW. [ONLINE] Available at: https://www.designprinciplesftw.com/collections/shneidermans-eight-golden-rules-of-interface-design. [Accessed 09 April 2018].

Software Testing Fundamentals. 2018. Unit Testing - Software Testing Fundamentals. [ONLINE] Available at: http://softwaretestingfundamentals.com/unit-testing/. [Accessed 16 April 2018].

Srivastava S, Kumari R, 2017. Machine Learning: A Review on Binary Classification. Internation Journal of Computer Applications, [Online]. Vol. 160, No. 7, 11-15. Available at: https://pdfs.semanticscholar.org/55b0/b0478fad2cd6c70d851e2be594a2788910ff.pdf [Accessed 10 April 2018].

SQL Injection Prevention Cheat Sheet - OWASP. 2018. SQL Injection Prevention Cheat Sheet - OWASP. [ONLINE] Available at: https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet. [Accessed 15 April 2018].

Techopedia.com. 2018. What is a Proof of Concept (POC)? - Definition from Techopedia. [ONLINE] Available at: https://www.techopedia.com/definition/4066/proof-of-concept-poc. [Accessed 17 April 2018].

Techopedia.com. 2018. What is Software Security? - Definition from Techopedia. [ONLINE] Available at: https://www.techopedia.com/definition/24866/software-security. [Accessed 15 April 2018].

The Dojo. 2018. The Mobile Test Pyramid | The Dojo. [ONLINE] Available at: https://dojo.ministryoftesting.com/dojo/lessons/the-mobile-test-pyramid. [Accessed 16 April 2018].

The Guardian. 2018. New Facebook controls aim to regulate political ads and fight fake news | Technology | The Guardian. [ONLINE] Available at: https://www.theguardian.com/technology/2018/apr/06/facebook-launches-controls-regulate-ads-publishers. [Accessed 07 April 2018].

tutorialspoint.com. 2018. UML - Activity Diagrams. [ONLINE] Available at: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm. [Accessed 10 April 2018].

tutorialspoint.com. 2018. Jackson Tutorial. [ONLINE] Available at: https://www.tutorialspoint.com/jackson/index.htm. [Accessed 11 April 2018].

Twitter4J - A Java library for the Twitter API. 2018. Twitter4J - A Java library for the Twitter API. [ONLINE] Available at: http://twitter4j.org/en/index.html. [Accessed 07 April 2018].

UML 2 Use Case Diagrams: An Agile Introduction. 2018. UML 2 Use Case Diagrams: An Agile Introduction. [ONLINE] Available at: http://www.agilemodeling.com/artifacts/useCaseDiagram.htm. [Accessed 09 April 2018].

Understanding WAR. 2018. Understanding WAR. [ONLINE] Available at: https://spring.io/understanding/WAR. [Accessed 11 April 2018].

UX courses. 2018. What is Human-Computer Interaction (HCI)? | Interaction Design Foundation. [ONLINE] Available at: https://www.interaction-design.org/literature/topics/human-computer-interaction. [Accessed 09 April 2018].

UX courses. 2018. User Interface Design Guidelines: 10 Rules of Thumb | Interaction Design Foundation. [ONLINE] Available at: https://www.interaction-design.org/literature/article/user-interface-design-guidelines-10-rules-of-thumb. [Accessed 09 April 2018].

UX courses. 2018. Shneiderman’s Eight Golden Rules Will Help You Design Better Interfaces | Interaction Design Foundation. [ONLINE] Available at: https://www.interaction-design.org/literature/article/shneiderman-s-eight-golden-rules-will-help-you-design-better-interfaces. [Accessed 09 April 2018].

Validation | Databases | ICT. 2018. Validation | Databases | ICT. [ONLINE] Available at: http://www.advanced-ict.info/databases/validation.html. [Accessed 09 April 2018].

VersionOne. 2018. Agile Programming Best Practices. [ONLINE] Available at: https://www.versionone.com/agile-101/agile-software-programming-best-practices/. [Accessed 20 April 2018].

Vox. 2018. Russia, Donald Trump, and “fake news”: How Russia pioneered the art of fake news - Vox. [ONLINE] Available at: https://www.vox.com/world/2018/4/5/17172754/russia-fake-news-trump-america-timothy-snyder. [Accessed 07 April 2018].

Weka 3 - Data Mining with Open Source Machine Learning Software in Java . 2018. Weka 3 - Data Mining with Open Source Machine Learning Software in Java . [ONLINE] Available at: https://www.cs.waikato.ac.nz/ml/weka/. [Accessed 07 April 2018].

What are HTML Web Templates? | HTML Design Website Templates. 2018. What are HTML Web Templates? | HTML Design Website Templates. [ONLINE] Available at: https://allwebcodesign.com/html-templates.htm. [Accessed 14 April 2018].

What is API - Application Program Interface? Webopedia. 2018. What is API - Application Program Interface? Webopedia. [ONLINE] Available at: https://www.webopedia.com/TERM/A/API.html. [Accessed 11 April 2018].

What is Validation in software testing? or What is software validation?. 2018. What is Validation in software testing? or What is software validation?. [ONLINE] Available at: http://istqbexamcertification.com/what-is-validation-in-software-testing-or-what-is-software-validation/. [Accessed 17 April 2018].

Will Anderson, Software validation techniques, Drug Information Journal 21 (1987), no. 4, 461–469.